Ein komponentenbasiertes Meta-Modell kontextabhängiger Adaptionsgraphen für mobile und ubiquitäre Anwendungen

Dissertation

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Thomas Springer
geboren am 20. Dezember 1972 in Bautzen

Gutachter: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Univ.-Prof. Dr.-Ing. habil. Klaus Meyer-Wegener
Prof. Dr.-Ing. habil. Sahin Albayrak

Dresden im Oktober 2004

DANKSAGUNG

Die vorliegende Arbeit entstand während meiner Tätigkeit als Stipendiat im Graduiertenkolleg „Werkzeuge zum effektiven Einsatz paralleler und verteilter Rechnersysteme“ sowie anschließend als Mitarbeiter am Lehrstuhl Rechnernetze der Technischen Universität Dresden. Zu ihrem gelingen haben viele Personen in ganz unterschiedlicher Weise beigetragen.

An erster Stelle möchte ich Herrn Prof. Alexander Schill danken, der einerseits durch das mir entgegengebrachte Vertrauen und die gewährten Freiheiten den Weg für das Gelingen der Arbeit geebnet und mir andererseits über die gesamte Bearbeitungszeit hinweg als kritischer Diskussionspartner und mit fachlichem Rat zur Seite stand. Besonders herzlich danke ich Herrn Prof. Klaus Meyer-Wegener und Prof. Sahin Albayrak für die freundliche Übernahme der Koreferate sowie zahlreiche konstruktive Diskussionen und kritische Anmerkungen, die den Inhalt der Arbeit wesentlich geprägt haben. Den Professoren, Stipendiaten und Kollegiaten des Graduiertenkollegs bin ich für zahlreiche Diskussionen, wertvolle Anregungen und Hinweise dankbar.

Den Mitarbeitern des Lehrstuhls Rechnernetze möchte ich für eine stets kollegiale und konstruktive Arbeitsatmosphäre danken. Stellvertretend nennen möchte ich Thomas Ziegert, der mich vor allem in der Anfangsphase bei der Themenfindung unterstützte und mir während der gesamten Bearbeitungszeit als Diskussionspartner zur Seite stand. Ebenso gilt mein Dank Steffen Göbel, Christoph Pohl und Steffen Zschaler für die ausdauernden und tiefgründigen Diskussionen sowie die aufmerksame Durchsicht der Entwürfe der Arbeit. Bedanken möchte ich mich auch bei Gerald Hübsch und Axel Spriestersbach für die gute Zusammenarbeit im gemeinsamen Projekt und für die Schaffung von Freiräumen während der Endphase der Arbeit.

Ganz herzlich danke ich meiner Frau Beate, meinem Sohn Janek sowie meinen Schwiegereltern für die moralische Unterstützung und das Schaffen der notwendigen Freiräume zum Schreiben der Arbeit.

Mein besonders herzlicher Dank gilt meiner Mutter, Frau Brigitte Springer, die mich auf meinem Weg stets unterstützt und damit das Gelingen dieser Arbeit ermöglicht hat.

ZUSAMMENFASSUNG

Gegenwärtige Infrastrukturen für verteilte Dienste und Anwendungen, insbesondere das Internet, entwickeln sich zunehmend zu mobilen verteilten Systemen. Durch die Integration drahtloser Netze, mobiler bzw. dedizierter Endgeräte und nicht zuletzt durch die Mobilität der Benutzer steigt die Heterogenität und Dynamik der Systeme hinsichtlich der eingesetzten Endgeräte, Kommunikationstechnologien sowie Benutzeranforderungen und Anwendungssituationen. Diese Eigenschaften sind mobilen Systemen inhärent und bleiben trotz der fortschreitenden Entwicklung der Technologien bestehen. Daraus resultieren spezifische Anforderungen an Anwendungen und Dienste, denen insbesondere die Softwareentwicklung Rechnung tragen muss.

In der vorliegenden Arbeit wird die Adaptivität von Softwaresystemen als wesentlicher Lösungsansatz für mobile verteilte Infrastrukturen thematisiert. Dazu werden wesentliche Mechanismen zur Adaption sowie der Überschneidungsbereich von Adaptionsmechanismen, „Context-Awareness“ und Softwareentwicklung untersucht. Ziel ist es, Erkenntnisse über Basismechanismen und Grundprinzipien der Adaption zu gewinnen und diese zur systematischen Entwicklung adaptiver Anwendungen auszunutzen.

Aus der Analyse des State-of-the-Art werden als erstes wichtiges Ergebnis der Arbeit wesentliche Basismechanismen zur Adaption identifiziert, umfassend klassifiziert und hinsichtlich eines Einsatzes in mobilen verteilten Infrastrukturen bewertet. Auf dieser Grundlage wird ein Meta-Modell zur systematischen Entwicklung adaptiver Anwendungen erarbeitet. Dieses erlaubt die Beschreibung adaptiver Anwendungen durch die Komposition von Basismechanismen zur Struktur- und Parameteradaption. Die Steuerung der Adaption durch Kontext und Meta-Informationen kann explizit beschrieben werden. Das Meta-Modell kann Entwickler beim Entwurf adaptiver Anwendungen unterstützen, stellt aber auch einen Ausgangspunkt für deren Analyse und Validierung sowie zur Kodegenerierung dar. Durch die explizite Beschreibung der verwendeten Adaptionsmechanismen und deren Abhängigkeiten von Kontext können Anwendungsmodelle außerdem zur Dokumentation verwendet werden. Im Rahmen der Validierung konnte die Integrierbarkeit der Basismechanismen und die flexible Anwendbarkeit des Modells zur systematischen Entwicklung adaptiver Anwendungen nachgewiesen werden.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

Kapitel 1

EINLEITUNG

1.1 Motivation

Im letzten Jahrzehnt hat sich das Internet zu einem bedeutenden Kommunikationsmedium unserer Zeit entwickelt. Dazu hat, neben Diensten wie E-Mail, News, File-Sharing, Telnet und IRC, vor allem das World Wide Web beigetragen, durch welches das Internet erstmals auch für eine kommerzielle Nutzung interessant wurde. Inzwischen ist das Internet zur bedeutendsten Plattform für verteilte Systeme, Dienste und Anwendungen geworden. Gegenwärtig etablieren sich neben den klassischen (nicht-zeitabhängigen) Diensten auch zeitabhängige und multimediale Dienste, wie Telefonie und Videokonferenzen. Außerdem hat mit der beständigen Entwicklung drahtloser Netzwerktechnologien und mobiler Endgeräte seit Beginn der 90er Jahre eine Integration mobiler Technologien in das Internet begonnen. Stationäre Rechner, vor allem Desktop PCs, mit einer in der Regel zuverlässigen Netzwerkverbindung und hoher Datenrate, bilden gemeinsam mit mobilen Geräten wie PDAs, Laptops und Mobiltelefonen mit überwiegend drahtlosen Verbindungen und geringer Datenrate das heutige Internet.

Gegenwärtige Aktivitäten in Forschung und Entwicklung werden diese Infrastruktur noch in vielfältiger Weise erweitern. So soll sich das zukünftige Internet zu einer Konvergenzplattform verschiedener Netze entwickeln (vom Telefonnetz bis hin zu traditionellen Datennetzen), die verschiedenste Kommunikations- und Datendienste integriert. Mit dem Internet2 wird eine breitbandige Festnetzinfrastruktur insbesondere auch für zeitabhängige multimediale Dienste entwickelt. Mobilfunknetze der 3. und 4. Generation zielen auf eine Erhöhung der verfügbaren Datenrate für die mobile Datenkommunikation sowie auf eine Integration verschiedener Zugangstechnologien in ein Backbone-Netzwerk. Durch die Verschmelzung des Internets mit digitalen Mobilfunksystemen soll zukünftig ein weitreichend verfügbares, mobiles Internet zur Verfügung stehen.

Im Forschungsbereich des Ubiquitous Computing wird ein Paradigmenwechsel in der Anwendung des Computers als Arbeitsgerät angestrebt. Kleinste, mit Sensoren ausgestatte Rechner können ihre Umgebung wahrnehmen, diese Informationen verarbeiten und miteinander kommunizieren. Dinge des alltäglichen Lebens werden durch die Integration solcher Kleinstrechner „smart“, d. h. sie können die Situation ihres Benutzers und dessen Umgebung wahrnehmen und sich an diese anpassen. Die Technik wird nach Mark Weiser [Wei91, Mat01] Mittel zum Zweck. Der PC als Universalwerkzeug wird durch vielfältige anwendungsspezifische Geräte und Rechner ersetzt werden, die nicht mehr den wesentlichen Teil der Aufmerksamkeit des Anwenders beanspruchen, sondern zum Großteil im Hintergrund arbeiten und somit eine weit bessere Konzentration auf die Erfüllung von Aufgaben ermöglichen. Neue Forschungsrichtungen, wie das Evernet [Ber00], werden die

Integration verschiedenster Dienste und Technologien weiter fortführen und für vielfältige Rechentechnik und Anwendungsgebiete eine permanente Verfügbarkeit von Kommunikationsverbindungen und Informationen ermöglichen. Begriffe wie „always on“, „ubiquitous“ und „information anywhere, anytime“ charakterisieren die Ziele dieser Bestrebungen.

Die Ausführungsumgebung für verteilte Dienste und Anwendungen wird sich somit zu einem mobilen verteilten System entwickeln. Durch die Integration drahtloser Netze, mobiler bzw. dedizierter Endgeräte und nicht zuletzt durch die Mobilität der Benutzer steigt die Heterogenität und Dynamik der Systeme. Damit eröffnen sich sowohl für Unternehmen als auch für private Anwender eine Fülle neuer Anwendungsfelder und Möglichkeiten, die insbesondere der steigenden Mobilität im täglichen Leben Rechnung tragen. Eine große Zahl von Mitarbeitern in vielen Unternehmen verbringt bereits heute einen großen Teil der Arbeitszeit außerhalb des Unternehmens, um z. B. vor Ort beim Kunden zu arbeiten. Für diese Mitarbeiter wird der mobile Zugriff auf die unternehmensinternen Anwendungen mehr und mehr zu einer Notwendigkeit. Vor allem moderne Unternehmen können von der Verfügbarkeit ihrer firmeninternen Dienste auf mobilen Endgeräten profitieren. Eine wesentliche Voraussetzung für die Benutzbarkeit, Akzeptanz und letztlich den kommerziellen Erfolg von Anwendungen und Diensten in einer solchen Infrastruktur ist die umfassende Unterstützung dieser Heterogenität und Dynamik. Diese ist vielfach in den heute eingesetzten Konzepten und Standards nicht oder nur bedingt vorhanden.

Traditionelle Infrastrukturen arbeiten in der Regel unter der Annahme, dass die Verfügbarkeit von Ressourcen für die Ausführung von Diensten gegeben ist. Als Endgeräte hatten bis vor einigen Jahren fast ausschließlich PCs oder Laptops mit PC-ähnlichen Eigenschaften Bedeutung. Deren Leistungsfähigkeit verdoppelt sich ebenso wie die von Rechnern der Festnetzinfrastruktur entsprechend des Moore’schen Gesetzes [Moo65] bei gleichem Preis in einem Zeitraum von 18 Monaten. Einen noch schnelleren Anstieg der Leistungsfähigkeit, vor allem der verfügbaren Datenrate, konnten drahtgebundene Netzwerke verzeichnen. Nach George Gilder [Gil00] erhöht sich die verfügbare Bandbreite in Kommunikationssystemen mindestens dreimal so schnell wie die verfügbare Rechenleistung.

Die Verfügbarkeit ausreichender Ressourcen schlug sich auch in der Anwendungsentwicklung nieder. Bei dieser stehen die Funktionalität sowie die Verteilung im Vordergrund. Internet-Anwendungen werden heute überwiegend als drei- oder mehrschichtige Anwendungen entworfen. Entsprechend dieser Architektur befindet sich der Großteil der Anwendungslogik im Festnetz. Clients werden als sogenannte „schlanke Clients“ entworfen, stellen aber vor allem für Anwendungen des E-Commerce wachsende Anforderungen an die Leistungsfähigkeit und multimedialen Fähigkeiten der Endgeräte. Demgegenüber verlagert das Konzept des abgekoppelten Arbeitens die benötigte Anwendungslogik auf das Endgerät. Damit verringert sich die Abhängigkeit von einer bestehenden Verbindung zum Festnetz, die Anforderungen an das Endgerät steigen aber entsprechend. Implizit setzt also der überwiegende Teil der heute verfügbaren Dienste und Anwendungen die Verfügbarkeit schneller und unterbrechungsfreier Kommunikationskanäle und/oder leistungsfähiger Endgeräte voraus. Unter anderem durch den Ansatz der „schlanken Clients“ werden ständig verfügbare Kommunikationsverbindungen notwendig. In ähnlicher Weise setzt der abgekoppelte Ansatz Ressourcen beim Endgerät voraus. Stehen die benötigten Ressourcen (z. B. bei Benutzung eines PDAs über eine GSM-Verbindung) nicht zur Verfügung, können Dienste und Anwendungen häufig nur sehr eingeschränkt bzw. gar nicht genutzt werden.

Mit der Entwicklung mobiler Technologien wandeln sich die Eigenschaften der Infrastruktur für verteilte Anwendungen. Die verfügbaren Ressourcen sind sehr heterogen und ändern sich sowohl von Benutzer zu Benutzer als auch innerhalb von Nutzersitzungen. Außerdem bringen viele mobile Technologien eine Begrenzung der verfügbaren Ressourcen mit sich. Die Unterschiede umfassen dabei in der Regel Größenordnungen. Beispielsweise steht mit GPRS für die Datenübertragung maximal 107,2 Kbit/s zur Verfügung (verfügbar sind in der Praxis ca. 50 Kbit/s), Gigabit Ethernet dagegen bietet eine Datenrate von 1 Gbit/s Rohdatenrate (theoretisch).

Die Eigenschaften der Heterogenität und Dynamik sind mobilen Systemen inhärent und bleiben trotz der fortschreitenden Entwicklung der Technologien bestehen [NoS95]. Obwohl beispielsweise die durchschnittlich verfügbare Datenrate steigen wird, werden die Unterschiede zwischen drahtgebundenen und drahtlosen Netzwerken auf absehbare Zeit bestehen bleiben oder sich weiter erhöhen. Durch mobile Infrastrukturen entstehen somit neben den neuen Anwendungsmöglichkeiten auch eine Reihe neuer Anforderungen. Diese sollen im Folgenden unter A1.1 bis A1.5 genannt werden:

A1.1. Heterogenität und Beschränkung der Ressourcen der Endgeräte

Tragbare und stationäre Rechner unterscheiden sich erheblich hinsichtlich ihrer Hard- und Softwareausstattung. Tragbare Rechner verfügen in der Regel aufgrund von Begrenzungen der Größe und Verfügbarkeit von Energie über geringere Ressourcen als stationäre Rechner. Dies umfasst zum einen Eigenschaften wie Rechenleistung, Speicherkapazität sowie Ein- und Ausgabegeräte, zum anderen Kriterien wie Betriebssystem, Browser, unterstützte multimediale Formate und die Funktionalität von Anwendungen. Demgegenüber sind tragbare Rechner zum Teil mit erweiterten oder speziellen Ressourcen ausgestattet, z. B. mit der Eingabemöglichkeit per Stift bei PDAs, die gegenwärtig auch für die Geräteklasse der Notebooks in Form von Tablet-PCs verfügbar wird. Eine Unterstützung der Heterogenität der Ressourcen muss sowohl die Eigenschaften der Anwendungsdaten als auch den Umfang der Anwendungsfunktionalität berücksichtigen. Ressourcenbeschränkungen sind dabei ebenso wie die Ausnutzung gerätespezifischer Ressourcen zu betrachten.

A1.2. Heterogenität der Zuverlässigkeit und Qualität von Netzwerkverbindungen

Durch den integrierten Einsatz vielfältiger drahtloser und drahtgebundener Netzwerktechnologien schwankt die Qualität der Kommunikationskanäle hinsichtlich Datenrate, Verzögerung, Jitter, Zuverlässigkeit, Fehlertoleranz und Kosten erheblich. Dies betrifft vor allem die Zugangsnetze und damit die sogenannte „letzte Meile“, d. h. den letzten Teil der Verbindung zwischen Netzwerkinfrastruktur und Endgerät. Zeitweilige Verbindungsunterbrechungen treten beim Einsatz mobiler Datenkommunikation um ein Vielfaches häufiger auf als bei der Nutzung einer reinen Festnetzinfrastruktur. Datenverluste, verbunden mit der Notwendigkeit einer erneuten vollständigen Übertragung, sind häufig die Folge. Außerdem bestehen aufgrund der Mobilität (z. B. zur Einsparung von Energie, zur Reduzierung von Kommunikationskosten bzw. durch Nichtverfügbarkeit einer Verbindung) häufig Phasen der Abkopplung vom Netzwerk. Heterogene Kommunikationstechnologien erfordern eine Unterstützung sowohl hinsichtlich der zu übertragenden Datenmenge als auch der Technologien zur Datenübertragung und –verteilung. Vor allem müssen die individuellen Merkmale der Netzwerke berücksichtigt werden.

A1.3. Heterogene Benutzeranforderungen und Kontext

Aus der Integration tragbarer, spezifischer Endgeräte und drahtloser Netzwerke in bestehende Infrastrukturen resultiert auch eine steigende Zahl von eingesetzten Geräten und

Benutzern. Dadurch gewinnen zum einen die Benutzerwünsche an Vielfalt, zum anderen erhöht sich auch die Menge möglicher Anwendungssituationen. Damit wird eine Personalisierung, d. h. die individuelle Anpassung von Diensten und Anwendungen an die Wünsche des jeweiligen Benutzers und den Benutzungskontext, insbesondere die jeweilige Anwendungssituation (Ort, Zeit, Aktivität des Benutzers, Rolle, usw.), erforderlich.

A1.4. Dynamik durch Mobilität

Der Einsatz mobiler Technologien bewirkt dynamische Veränderungen in der bisher statischen Topologie des Internets. Geräte, Verbindungen und Ressourcen können jederzeit hinzugefügt bzw. entfernt werden. Die Zahl der verwendeten Geräte und Benutzer schwankt dabei erheblich zwischen Zeiten intensiver und geringerer Dienstnutzung. So können Benutzer auch während einer Anwendungssitzung ihre Ein- und Ausgabemethode, die Netzwerktechnologie oder das Endgerät wechseln. Außerdem kann ein Benutzer auch mehrere Endgeräte zur gleichen Zeit einsetzen, etwa um mit einem PDA per Stift Daten einzugeben und gleichzeitig Ausgaben an einem großen Bildschirm anzusehen (Multimodality). Aus der Sicht der mobilen Teilnehmer variiert als Folge der Mobilität außerdem das Dienst- und Ressourcenangebot. Damit wird eine dynamische Verwaltung von Topologieinformationen, Systemressourcen und Diensten notwendig. Annahmen über die Verfügbarkeit bestimmter Ressourcen und Dienste treffen in der Regel nicht mehr zu, vielmehr müssen Systeme und Anwendungen in der Lage sein, zur Laufzeit auf Änderungen der Ausführungsumgebung zu reagieren.

A1.5. Sicherheit

Mobile Technologien werfen auch spezielle Probleme in Bezug auf Sicherheit auf. Mobile Endgeräte können wesentlich einfacher gestohlen werden, gehen verloren oder werden zerstört. Des Weiteren können aufgrund der beschränkten Ressourcen, insbesondere der Verarbeitungskapazität, Konzepte und Algorithmen zur Durchsetzung von Schutzzielen nur in begrenztem Umfang eingesetzt werden. Drahtlose Kommunikationsverbindungen könenn leichter abgehört werden als drahtgebundene. Aufgrund der Mobilität besteht außerdem die Notwendigkeit der Authentisierung und Autorisierung mobiler Nutzer. Deshalb sind Sicherheitsmechanismen mit geringen Ressourcenanforderungen sowie erweiterte und neue Sicherheitskonzepte erforderlich. Für die weiteren Betrachtungen im Rahmen der vorliegenden Arbeit wird die Problematik der Sicherheit jedoch ausgeschlossen.

1.2 Problemstellung und verwandte Arbeiten

Nach [Ste99] entspricht ein Gesamtsystem seinen Anforderungen mit einer gewissen Güte, der Dienstgüte. Um eine gewisse Dienstgüte zu erreichen, müssen die dafür notwendigen Ressourcen (z. B. Speicherplatz, Bandbreite, Prozessorleistung) zur Verfügung stehen. Grundsätzlich kann auf eine begrenzte oder sich dynamisch ändernde Ressourcenverfügbarkeit auf zwei Arten reagiert werden:

1. *Ressourcenreservierung*: Eine gleichbleibende Dienstgüte wird durch die Reservierung der notwendigen Ressourcen garantiert. Die Reservierung erfolgt in der Regel über die gesamte Laufzeit, nachträglich verfügbare Ressourcen können somit nicht dynamisch einbezogen werden. Vielmehr müssen die notwendigen Ressourcen zum Zeitpunkt des Dienststarts zur Verfügung stehen, andernfalls wird die Anforderung abgewiesen. Der Dienst wird somit entweder in der geforderten Dienstgüte oder gar nicht erbracht. Au-

ßerdem bleiben als Folge möglicher Überreservierungen Ressourcen ungenutzt, wodurch sich die Auslastung der verfügbaren Ressourcen verschlechtert.

2. *Adaption (Skalierung)*: Eine variable Dienstgüte wird durch die Anpassung des Dienstes an die verfügbaren Ressourcen erreicht. Damit kann eine bestimmte Dienstgüte nicht zugesagt werden. Zur Laufzeit frei werdende Ressourcen können jedoch dynamisch einbezogen werden, um die Dienstgüte zu erhöhen. Dienste können aber auch dann erbracht werden, wenn die maximal vom Dienst benötigten Ressourcen nicht zur Verfügung stehen.

In Abbildung 1-1 ist der Zusammenhang zwischen Reservierung und Adaption dargestellt. Die Ausführungsumgebung verfügt nur über eine begrenzte Menge von Ressourcen. Diese müssen außerdem zum Teil fair auf mehrere Anwendungen bzw. Anwender aufgeteilt werden (z. B. Kommunikationsnetzwerk, Rechenleistung). Die von einem Dienst benötigten Ressourcen werden durch die vom Entwickler festgelegte Dienstgüte bestimmt. Dabei sind verschiedene Ressourcenbelegungen für eine bestimmte Dienstgüte denkbar. Beispielsweise kann für die Kodierung eines Bildes oder Videos durch unterschiedliche Kompressionsverfahren bei gleicher Qualität mehr Rechenleistung und weniger Speicher bzw. weniger Rechenleistung und entsprechend mehr Speicher verwendet werden. Sind mindestens die Ressourcen für die vorgesehene Dienstgüte in der Ausführungsumgebung verfügbar, kann der Dienst in dieser erbracht werden. Durch Reservierung der Ressourcen kann die Dienstgüte außerdem zugesichert werden. Weiterhin besitzt jeder Dienst ein Minimum an benötigten Ressourcen, unterhalb dessen der Dienst nicht mehr sinnvoll erbracht werden kann. Beispielsweise existiert ein Minimum für die Größe von Bildern unterhalb dessen der Inhalt eines Bildes nicht mehr erfassbar ist. Befinden sich die verfügbaren Ressourcen im Bereich zwischen dem geforderten und dem minimalen Ressourcenbedarf eines Dienstes, kann der Dienst nur noch durch Adaption erbracht werden. Dies ist durch die begrenzte Leistungsfähigkeit mobiler Technologien in mobilen verteilten Systemen in der Regel der Fall. Die Ressourcenreservierung kann deshalb in mobilen Infrastrukturen nur sehr begrenzt angewendet werden. In der Regel muss eine Adaption erfolgen, um traditionelle und neue Dienste auch in einem mobilen Umfeld erbringen zu können.

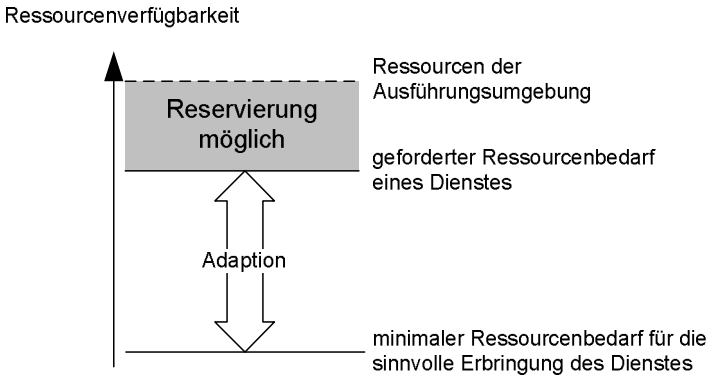


Abbildung 1-1: Ressourcenreservierung und Adaption

Adaption stellt also einen allgemeinen Lösungsansatz für die Anforderungen heterogener und dynamischer Infrastrukturen dar (siehe Anforderungen A1.1 bis A1.4). Seit Beginn der 90er Jahre ist Adaption Thema zahlreicher Forschungsprojekte im Bereich Mobile Computing. Die Konzepte verfolgen dabei zwei grundlegende Ansätze. Zum einen werden Lösungen für spezifische Problemstellungen vertiefend untersucht (siehe Abschnitt 2.1). Die Konzepte umfassen ein abgekoppeltes Arbeiten auf dem Endgerät, ein autonomes Arbeiten

im Festnetz, Adaptionsfunktionen sowohl auf dem Endgerät als auch im Festnetz sowie angepasste Protokolle und Mechanismen zur Unterstützung der Kommunikation über drahtlose bzw. heterogene Netzwerkverbindungen. Zum Teil wurden einzelne Mechanismen auch kombiniert, um spezifische Endgeräte (z. B. Palm PDAs) bzw. charakteristische Zustände der Ausführungsumgebung zu unterstützen (z. B. ein abgekoppeltes Arbeiten).

Zum anderen wurden Laufzeitumgebungen untersucht, die Adaptionsmechanismen als Komponenten betrachten und eine Middleware zur Ausführung von Adaptionskomponenten bereitstellen. Damit gekoppelt wurden zum Teil Algorithmen entwickelt, die eine automatische Erzeugung von Konfigurationen von Komponenten ermöglichen (siehe Abschnitt 2.2). Diese legen adaptiven Komponentensystemen die Struktur von Pfaden bzw. gerichteten azyklischen Graphen zugrunde. Auf der Grundlage des Architekturmusters „Pipes-and-Filters“ werden die einzelnen Mechanismen zu Pfaden bzw. Graphen zusammengesetzt. Dabei hat sich vor allem die durch das Muster vorgegebene lose Kopplung als vorteilhaft erwiesen. Schwerpunkt dieser Lösungen sind aber vor allem Probleme der Laufzeitplattformen [GWB+01, YWR+99, KiF00], bestimmte Mechanismen der Adaption (z. B. Protokolladaption [SuB01, MHM+98] oder Medienkonvertierung [Mar01b]) oder Aspekte der automatischen Pfad- bzw. Graphenerzeugung [CRS98, Mar01b].

Einen weiteren Einfluss auf die Betrachtung der Adaption haben Ansätze und Lösungen aus dem Forschungsbereich Context-Awareness. In diesem werden die Gewinnung, Verarbeitung und Bereitstellung von Kontext sowie dessen Einsatz in kontextabhängigen Anwendungen untersucht. Der Schwerpunkt liegt auf personenbezogenen und weiteren nichttechnischen Informationen (z. B. der Aufenthaltsort und die Aktivität des Benutzers), die eine Unterstützung der Interaktionen zwischen Mensch und Anwendung ermöglichen. Technische Informationen über die Ausführungsumgebung und eine Anpassung von Anwendungen an diese, werden im Bereich Context-Awareness nicht vertiefend untersucht. Auf der anderen Seite stellen auch wenige der unter den ersten beiden Punkten genannten Systeme einen expliziten Bezug zu Kontext her bzw. verwenden einen Kontextdienst als Informationsquelle für das Auslösen und die Steuerung von Adaptionsoperationen.

Zusammenfassend lässt sich feststellen, dass vielfältige Adaptionsmechanismen bereits untersucht wurden. Ebenso existieren bereits Laufzeitplattformen, die eine Basis zur Kombination dieser Adaptionsmechanismen liefern. Der überwiegende Teil der Ansätze beschreibt jedoch Lösungen zur Unterstützung bestimmter Teilprobleme mobiler verteilter Infrastrukturen bzw. betrachtet nur eine Teilmenge der verfügbaren Mechanismen. Eine systematische Entwicklung adaptiver Anwendungen durch die Kombination und Wiederverwendung bereits verfügbarer Mechanismen wurde jedoch nicht betrachtet. Eine Klassifikation von Adaptionsmechanismen erfolgte nur sehr grob in systembasierte und anwendungsbasierte Mechanismen [Sat96] bzw. anhand von vier Ebenen der Systemarchitektur [FDB+99]. Ebenso fehlt eine explizite Beschreibung der Zusammenhänge zwischen Zuständen der Ausführungsumgebung und einer davon abhängigen Auswahl bzw. Konfiguration von Adaptionsmechanismen. Lösungen zur Pfad- bzw. Graphenerzeugung ermöglichen nur die Auswahl von Mechanismen, nicht jedoch deren Konfiguration oder die dynamische Änderung der Eigenschaften von bereits instanziierten Pfaden bzw. Graphen.

1.3 Ziele der Arbeit

In der vorliegenden Arbeit sollen wesentliche Mechanismen zur Adaption sowie der Überschneidungsbereich von Adaption und „Context-Awareness“ untersucht werden. Ziel ist

es, Erkenntnisse über Basismechanismen und Grundprinzipien der Adaption zu gewinnen und diese zur systematischen Entwicklung adaptiver Anwendungen auszunutzen.

Dazu soll der State-of-the-Art des Forschungsbereiches Mobile und Ubiquitous Computing hinsichtlich grundlegender Prinzipien und wiederverwendbarer Mechanismen analysiert werden. Ziel ist die Identifizierung und Klassifizierung von Basismechanismen, die als Grundbausteine adaptiver Anwendungen eingesetzt werden können. Dazu muss ein geeignetes Klassifikationsschema entwickelt werden. Außerdem soll der Überschneidungsbereich von Adaption und Context-Awareness untersucht werden, um die Verwendbarkeit von Kontext zur Steuerung von Adaptionsprozessen zu prüfen. Die gewonnenen Erkenntnisse sollen bei der Erarbeitung eines Meta-Modells zur Beschreibung adaptiver Anwendungen eingesetzt werden. Die grundlegende Struktur des Meta-Modells bilden Adaptionsgraphen. Die Knoten repräsentieren die Basismechanismen zur Adaption. Die Kanten beschreiben die Interaktionen zwischen den einzelnen Mechanismen. Weitere Elemente des Meta-Modells sollen die explizite Beschreibung der Zugriffe auf Kontext sowie dessen Abbildung auf Adaptionsinformationen ermöglichen.

Das Meta-Modell soll Entwickler vor allem beim Entwurf adaptiver Anwendungen unterstützen, stellt aber auch einen Ausgangspunkt für deren Analyse und Validierung sowie zur Codegenerierung dar. Das Meta-Modell beschreibt adaptive Anwendungen dabei aus der Sicht der Adaption. Diese Sicht kann mit weiteren Sichten kombiniert werden, um vollständige Anwendungen zu modellieren. Durch die explizite Beschreibung der verwendeten Adaptionsmechanismen und deren Abhängigkeiten von Kontext können Anwendungsmodelle außerdem zur Dokumentation verwendet werden.

Im Rahmen der Arbeit wird die Adaption klassischer Internetanwendungen wie E-Mail, Instant Messaging, FTP und WWW betrachtet. Diese greifen wiederholt auf gleichartige Anwendungsdaten bzw. Medien zu. Aus dem Zugriffsverhalten resultieren aperiodische und unzusammenhängende Datenströme. Außerdem erfolgt eine Konzentration auf diskrete Medien. Diese Anwendungsklasse wird gegenwärtig für tragbare Rechner erschlossen. Vor allem E-Mail, webbasierte Anwendungen und in letzter Zeit auch Instant Messaging sind für den Einsatz in Unternehmen von großer Bedeutung. Nachfolgend bezieht sich der Ausdruck adaptive Anwendung auf die genannte Anwendungsklasse.

Für die vorliegende Arbeit werden folgende Thesen formuliert:

These 1: Es existieren wiederverwendbare Basismechanismen zur Adaption von Anwendungen an die Heterogenität und Dynamik mobiler verteilter Infrastrukturen.

These 2: Adaptive Anwendungen können durch die Kombination von Basismechanismen systematisch entwickelt und realisiert werden.

These 3: Informationen über die Ausführungsumgebung und die Anwendungssituation sowie Benutzereinstellungen können in Form von Kontext zur Steuerung der Adaption verwendet werden.

1.4 Begriffsbestimmung

Der Begriff Adaption besitzt in verschiedenen Zusammenhängen eine unterschiedliche Bedeutung. Im technischen Sinne wird unter Adaption ein Ausgleich zwischen nicht zusammenpassenden Systemen verstanden. So ist ein Adapter ein Zwischenstück zum Verbinden von an sich unvereinbaren Geräteteilen [Lan89]. Allgemein kann Adaption wie folgt definiert werden:

*„**Adaptation**: [lateinisch] auch Adaption, 1. Anpassung, Anpassungsvermögen 2. (biol., med.) Anpassung an Umweltbedingungen (Organe) 3. Bearbeitung, Umarbeitung eines Kunstwerks, Neubearbeitung in einer anderen Gattung“ [Lan89].*

Adaption bezeichnet also eine Aktivität in Reaktionen auf Veränderungen. Wesentlich sind die Fragestellungen Was? und Woran? angepasst wird. Diese bezeichnen den Gegenstand und das Ziel einer Anpassung (siehe Abbildung 1-2).

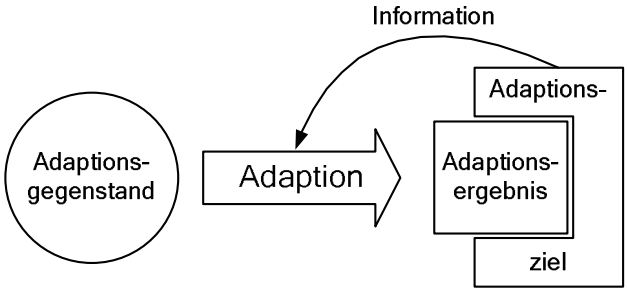


Abbildung 1-2: Bestandteile eines Adaptionsprozesses

Das in der vorliegenden Arbeit betrachtete *Adaptionsziel* sind gemäß der Anforderungen A1.1 bis A1.4 die Ressourcenengpässe mobiler verteilter Infrastrukturen sowie Benutzeranforderungen und die Anwendungssituation. Wie beschrieben, sind Ressourcen innerhalb der Festnetzinfrastruktur in der Regel in ausreichendem Maß vorhanden bzw. können kostengünstig beschafft werden. Beschränkungen resultieren hauptsächlich aus den Eigenschaften drahtloser Zugangsnetzwerke und mobiler Endgeräte. *Adaptionsgegenstand* sind die Daten, die Kommunikation sowie die Struktur mobiler und ubiquitärer Anwendungen.

Adaption wird in den weiteren Ausführungen als Anpassung der Anwendungsdaten, deren Übertragung sowie der Anwendungsstruktur an die verfügbaren Ressourcen der Zugangsnetzwerke und Endgeräte sowie an Benutzereinstellungen und die Anwendungssituation verstanden.

Kontext

Eine notwendige Voraussetzung für die Adaption ist es, dass Informationen über das Adaptionsziel verfügbar sind bzw. ermittelt werden können, um Adaptionsprozesse steuern zu können. Die Gewinnung, Verarbeitung und Speicherung von Informationen über die Ausführungsumgebung sind wesentliche Ziele des Forschungsbereiches Context-Awareness. Die betrachteten Informationen werden unter dem Begriff Kontext zusammengefasst, der durch die folgende Definition näher bestimmt wird:

„Context is any Information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application.” [Dey01]

Demnach ist Kontext jede für die Charakterisierung der Situation eines Entities nutzbare Information. Diese können in Form von Ein- und Ausgabedaten von Anwendungen explizit vorliegen. Der überwiegende Teil der Informationen muss jedoch in irgendeiner Weise gewonnen werden. Context-Awareness betrachtet vor allem letzteren Teil der Informationen („ ... all but the explicit input and output of an application” [LiS00]). Ziel ist es,

implizte Informationen explizit verfügbar zu machen. Im Rahmen dieser Arbeit werden in Bezug auf Kontext die folgenden Begriffe verwendet:

- Ein *Kontextwert* repräsentiert eine einzelne Information. Die Menge aller Kontextwerte bildet den Kontext.
- Jeder Kontextwert besitzt einen Typ, der als *Kontexttyp* bezeichnet wird.
- Eine Verarbeitung ist *kontextabhängig*, wenn sie die Fähigkeit zur Wahrnehmung von Kontext besitzt und ein definierter Zusammenhang zwischen Kontextwerten und Verarbeitungsergebnissen besteht, d. h. sich in Reaktion auf Änderungen des Kontextes die Verarbeitungsergebnisse in definierter Weise ändern.
- Ein *Kontextdienst* ist ein Dienst, der aus Sicht seiner Benutzer Kontext erfasst, verarbeitet und speichert und diesen den Benutzern über eine Anfrageschnittstelle zur Verfügung stellt.

In Bezug auf Interaktionen mit einem Kontextdienst können zwei Rollen definiert werden:

1. *Kontextbenutzer* interagieren über die Abfrageschnittstelle mit dem Kontextdienst, um Kontext zu ermitteln. Kontextbenutzer sind damit die „Verbraucher“ von Kontext.
2. *Kontextlieferanten* interagieren mit dem Kontextdienst, um Kontext zur Verfügung zu stellen. Kontextlieferanten sind also die „Erzeuger“ von Kontext.

Bestandteile eines Systems oder einer Anwendung können eine der beiden Rollen aber auch beide Rollen gleichzeitig einnehmen. Beispielsweise kann eine Anwendung sowohl Kontextbenutzer (wenn sie Informationen vom Kontextdienst abfragt) als auch Kontextlieferant sein (z. B. wenn sie Informationen über Benutzereigaben ermittelt und dem Kontextdienst zur Verfügung stellt).

Adaptive Anwendung

Anwendungen können auf Informationen über ihre Ausführungsumgebung in Form von Kontext zugreifen und sich dadurch an diese anpassen. Während der Begriff „adaptiert“ einen Zustand der Anpassung im Sinne einer Spezialisierung an eine bestimmte Ausführungsumgebung beschreibt, bezeichnet die *Fähigkeit zur Adaption* die Eigenschaft der dynamischen Anpassung einer Anwendung zur Laufzeit.

Adaptive Anwendungen werden durch die Fähigkeit zur dynamischen Adaption von Anwendungsdaten, der Kommunikation sowie der Anwendungsstruktur in Reaktion auf Änderungen des Kontextes zur Laufzeit charakterisiert. Kontext bezeichnet die verfügbaren Ressourcen der Zugangsnetzwerke und Endgeräte sowie Benutzereinstellung und die jeweilige Anwendungssituation. Eine notwendige Voraussetzung dafür ist die Verfügbarkeit des Kontextes.

Ein *Adaptionsmechanismus* ist ein Mechanismus, der in Bezug auf einen bestimmten Zustand der Ausführungsumgebung eine an diesen Zustand angepasste Verarbeitung realisiert bzw. deren Realisierung ermöglicht.

Adaptionsgraphen sind gerichtete Graphen und können Zyklen enthalten. Komponenten und Konnektoren stellen die Knoten, die Verknüpfungen zwischen Ports und Rollen die Kanten eines Adaptionsgraphen dar.

Meta-Modell

Die OMG-Terminologie unterscheidet vier Modellebenen. Diese sind: Meta-Meta-Modell (M3), Meta-Modell (M2), Modell (M1) und Instanz von Modell (M0). Die Meta-Object Facility (MOF) ist eine abstrakte Sprache zur Spezifikation von Meta-Modellen und damit auf Ebene M3 angesiedelt. Elemente von Meta-Modellen sind der Ebene (M2) zugeordnet und ermöglichen die Beschreibung von Anwendungsmodellen (M1), deren Instanzen lauffähige Anwendungen sind (M0).

1.5 Gliederung der Arbeit

In der Arbeit wurden zunächst die Begriffe Adaption, adaptive Anwendung, Adaptionsgraph, Kontext und Meta-Modell bestimmt. Im Folgenden werden Forschungsarbeiten aus den Bereichen Mobile und Ubiquitous Computing, Context-Awareness und Softwaretechnologie untersucht. Als erstes wesentliches Ergebnis dieser Arbeit werden wiederverwendbare Basismechanismen zur Adaption identifiziert. Außerdem werden Schemen zur Klassifikation und Bewertung der Basismechanismen erarbeitet. Auf Basis der klassifizierten Basismechanismen und weiterer Erkenntnisse über die Prinzipien der Adaption wird ein Meta-Modell entwickelt, das eine Komposition der Basismechanismen zu adaptiven Anwendungen ermöglicht. Weitere Elemente des Meta-Modells dienen gemäß der Zielstellung einer expliziten Beschreibung der Verwendung von Kontext zur Steuerung von Adaptionsmechanismen. Die flexible Einsetzbarkeit für die Entwicklung, Beschreibung, Analyse und Dokumentation adaptiver Anwendungen wird anhand vielfältiger Szenarien sowohl aus der Literatur als auch aus eigener Entwicklung validiert. Abgeschlossen wird die vorliegende Arbeit mit einer Zusammenfassung der Ergebnisse sowie einem Ausblick auf weitere Forschungsmöglichkeiten.

Kapitel 2

VERWANDTE ARBEITEN

Mechanismen zur Unterstützung heterogener und dynamischer Infrastrukturen werden seit der Verwendung von Rechnernetzen und verteilten Systemen untersucht und entwickelt. Mit der Integration mobiler Geräte und drahtloser Netztechnologien in verteilte Infrastrukturen wurde die Heterogenität dieser Systeme nochmals signifikant erhöht. In den Forschungsgebieten der verteilten Systeme sowie des Mobile und Ubiquitous Computing wurden deshalb Lösungsansätze entwickelt, die die Mobilität, Heterogenität und Dynamik von Infrastrukturen thematisieren. Im vorliegenden Kapitel werden verwandte Arbeiten verschiedener Forschungsgebiete beschrieben, die Einfluss auf den in dieser Arbeit entwickelten Lösungsansatz genommen haben. Die Konzepte werden in fünf Schwerpunktbereiche unterteilt. Jeder der Schwerpunkte wird einleitend in Bezug zur Lösung gesetzt und abschließend zusammenfassend dargestellt. Damit können die Teile des Kapitels auch unabhängig voneinander gelesen werden.

Im ersten Teil des Kapitels werden zunächst einzelne Mechanismen und Ansätze zur Anpassung an die heterogene und dynamische Laufzeitumgebung identifiziert und erläutert. Der zweite Teil untersucht die Ansätze zur Komposition von Mechanismen zu Komponentenpfaden oder –graphen. Diesen liegt überwiegend die Idee einer losen Kopplung autonomer Komponenten ähnlich dem „Pipes und Filters“ Konzept zugrunde. Im dritten Teil werden Architekturen zur Realisierung adaptiver Anwendungen insbesondere hinsichtlich einer Verteilung und freien Platzierbarkeit der Komponenten untersucht. Die Platzierung und deren Änderung in Abhängigkeit von den aktuell verfügbaren Ressourcen stellen ebenfalls wesentliche Mechanismen zur Adaption von Anwendungen dar. Im vierten Teil werden die Aspekte des Kontextbezuges näher untersucht. Damit wird ein Zusammenhang zwischen dem Forschungsgebiet „Context-Awareness“ und Adaption hergestellt. Kontext beschreibt die aktuelle Umgebung einer Anwendung und des Benutzers und liefert so die Informationen, die für die Steuerung der Adaption notwendig sind. Im fünften Teil werden Konzepte und Sprachen zur Beschreibung adaptiver Anwendungen sowie von Aspekten der Adaption vorgestellt. Im Mittelpunkt stehen insbesondere die Möglichkeiten einer abstrakten Beschreibung von Adaptionsmechanismen in Form von Softwarekomponenten und eine Komposition von Mechanismen zu sogenannten Adaptionspfaden bzw. -graphen.

2.1 Adaptionsmechanismen zur Unterstützung mobiler Infrastrukturen

Der Forschungsbereich „Mobile Computing“ hat die Integration tragbarer Rechner (vorrangig Laptops) in bestehende Rechnernetze zum Ziel. Seit Beginn der 90er Jahre werden Mechanismen entwickelt, die das Arbeiten mit tragbaren Rechnern und drahtlosen Kommunikationsverbindungen unterstützen. Erste Forschungsarbeiten behandelten vor allem

Mechanismen, die transparent für Programmierer und Anwendung arbeiteten. Untersucht wurde zunächst die Problematik des abgekoppelten Arbeitens mit vergleichsweise leistungsfähigen tragbaren Rechnern (in der Regel Laptops). Dabei wurden vor allem die Probleme häufiger Verbindungsunterbrechungen sowie der Abkopplung über einen längeren Zeitraum (z. B. in Folge der Mobilität des Benutzers) adressiert (siehe Abschnitt 2.1.1).

Weitere bedeutende Problemstellungen kamen später hinzu. Dies sind die geringe bzw. stark variierende Datenrate sowie hohe Antwortzeiten infolge des Einsatzes drahtloser Netzwerktechnologien und der Mobilität (die gilt jedoch auch für Modem- (geringe Datenrate) und Satellitenverbindungen (hohe Antwortzeit)). Forschungsaktivitäten in diesem Bereich untersuchten vor allem Mechanismen zur Reduzierung der zu übertragenden Datenmenge durch Manipulation der Anwendungsdaten (siehe Abschnitt 2.1.3) sowie zum Vorabladen bzw. Verzögern von Daten (siehe Abschnitt 2.1.1). Entsprechende Mechanismen können außerdem zur Anpassung von Anwendungen an die begrenzten Ressourcen tragbarer Endgeräte (d. h. Speicher, Ausgabemöglichkeiten und unterstützte multimediale Datenformate) eingesetzt werden (siehe Abschnitt 2.1.3).

Die Heterogenität der eingesetzten Kommunikationstechnologien machte außerdem neue bzw. erweiterte Kommunikationsprotokolle notwendig, die die Besonderheiten der drahtlosen Technologien unterstützten. Beispielsweise wurden in Protokollen für drahtgebundene Netze zum Teil Annahmen getroffen, die für drahtlose Netze nicht zutreffen und deren Leistung und Dienstgüte erheblich vermindern [CaI95, BPS+97a]. Insbesondere zur Fehlerbehandlung wurden neue Mechanismen benötigt, da Paketverluste in drahtgebundenen Netzen im wesentlichen auf Stausituationen zurückgeführt werden können, in drahtlosen Netzwerken treten diese jedoch auch gehäuft durch Störungen während der Übertragung auf. Die z. B. in TCP [Pos81, Jac88] eingesetzten Mechanismen zur Behandlung von Paketverlusten, d. h. Stausituationen, sind auf drahtlose Netzwerke nur mit Leistungsverlusten anwendbar [CaI95] und machen Protokollerweiterungen notwendig. Ansätze zu dieser Problemstellung werden in Abschnitt 2.1.2 diskutiert.

2.1.1 Abgekoppelte und autonome Operationen

Eines der bedeutendsten Probleme, das sich mit der Benutzung mobiler Endgeräte und drahtloser Verbindungen ergibt, ist die Unterbrechung der Verbindung zwischen Endgerät und Festnetz über einen längeren Zeitraum. Diese Abkopplung kann vorhersagbar und nicht-vorhersagbar erfolgen. Unterbricht der Benutzer z. B. die Verbindung, um Energie oder Verbindungskosten zu sparen, ist die Abkopplung vorhersagbar. Nicht-vorhersagbare Abkopplungen resultieren unter anderem aus der Bewegung in einen Funkschatten, Funkstörungen, Fehlern im Zugangsrechner, einer Überlastung des Netzwerkes oder von Serverrechnern.

Zur Lösung des Problems können zwei generelle Ansätze unterschieden werden. Der erste Ansatz wurde unter dem Begriff *abgekoppelte Operationen* erstmals mit dem verteilten Dateisystem Coda vorgestellt [KiS92, SKM+93, Kis96]. Ziel dieses ist es, während einer bestehenden Verbindung die Voraussetzung für ein potentiell abgekoppeltes Arbeiten zu schaffen, d. h. die für die zukünftige Arbeit notwendigen Daten und die zugehörigen Verarbeitungsfunktionen auf dem Endgerät verfügbar zu halten. Der zweite Ansatz der *autonomen Operationen* unterstützt die Ausführung von Programmteilen auf Rechnern im Festnetz unabhängig von einer bestehenden Netzwerkverbindung zum Endgerät und insbesondere unabhängig vom Benutzer. Ergebnisse der autonomen Verarbeitung können nach Aufbau einer neuen Verbindung zum Endgerät übermittelt werden. Abgekoppelte Operationen stellen somit eine Untermenge der autonomen Operationen dar, da die Abarbeitung

der Programmteile auf beliebigen Rechnern und somit auch auf dem Endgerät erfolgen kann. Während abgekoppelte Operationen vor allem die lokale Verfügbarkeit von Daten und die Herstellung der Konsistenz nach dem Wiederherstellen der Verbindung betrachten, steht bei autonomen Operationen die dynamische Platzierung bzw. Migration von Programmcode sowie der zu verarbeitenden Daten im Vordergrund. Nachfolgend werden grundlegende Mechanismen für beide Operationsarten beschrieben.

2.1.1.1 Abgekoppelte Operationen

In [FiG94] wird eine allgemeine Entwurfsmethode für abgekoppelte Operationen vorgestellt. Diese basiert auf der Einführung einer Stellvertreterkomponente auf dem Rechner des Clients. Die Komponente arbeitet in einem der beiden Zustände „verbunden“ oder „nicht-verbunden“ (siehe Abbildung 2-1). Für jeden der Zustände enthält sie Implementierungen der Funktionen der Serverschnittstelle, d. h. für jede Funktion f() der Serverschnittstelle, enthält der Stellvertreter die Funktionen f_connected() und f_disconnected(). Funktionen eines Zustandes dürfen dabei nicht von Funktionen oder Zustandsinformationen eines anderen Zustandes abhängen. Die einzelnen Zustände des Stellvertreters werden durch Übergangsfunktionen ineinander überführt. Dabei werden alle für den Zustandsübergang notwendigen Aktionen innerhalb der jeweiligen Übergangsfunktion gekapselt.

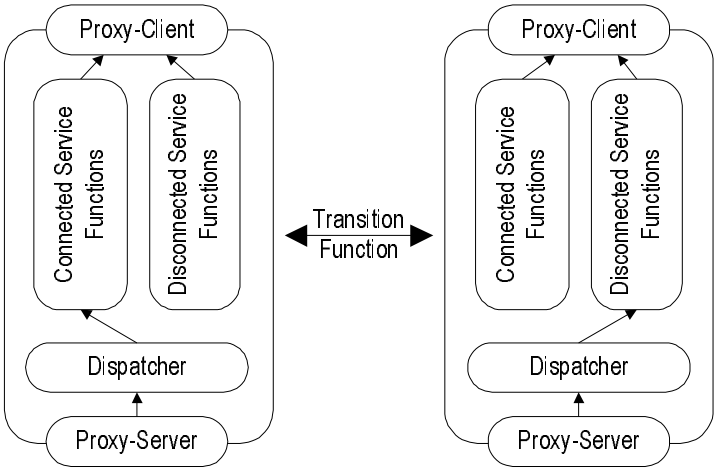


Abbildung 2-1: Stellvertreter im Zustand „verbunden” (links) und „nicht-verbunden” (rechts) [FiG94]

Über die Komponente Proxy-Server bietet der Stellvertreter die Schnittstelle des eigentlichen Anwendungsservers an und nimmt Aufrufe von Clients an den Server entgegen. Der Dispatcher vermittelt die Aufrufe in Abhängigkeit des Zustandes, in dem sich der Stellvertreter befindet, an die entsprechende Implementierung der aufgerufenen Funktion. Im Zustand „verbunden“ werden alle Aufrufe über die Komponente Proxy-Client an den Server weitergeleitet. Dies ermöglicht die Beobachtung der Aufrufe des Clients und die Vorbereitung der Anwendung für den Zustand „nicht-verbunden“. In diesem Zustand emulieren die im Stellvertreter enthaltenen Funktionen die Funktionen des Servers.

Abgekoppelte Operationen stellen somit keinen atomaren Mechanismus dar, sondern werden durch eine Kombination mehrerer Mechanismen realisiert. Während der Abkopplung muss die Anwendung unabhängig von einer bestehenden Netzwerkverbindung arbeiten können. Die lokale Verfügbarkeit der zu verarbeitenden Daten sowie der Verarbeitungsfunktionen ist dafür eine notwendige Voraussetzung. Die Funktionen einer Anwendung müssen somit in unabhängig von einer bestehenden Netzwerkverbindung arbeitende

Verarbeitungsfunktionen sowie Funktionen zur Wiederherstellung der Konsistenz überführt werden. Außerdem müssen die Funktionen statisch (z. B. in Form eines Stellvertreters) bzw. dynamisch (z. B. durch mobilen Code) auf dem Endgerät installiert werden. Analog dazu müssen die Anwendungsdaten, die während der Abkopplung bearbeitet werden sollen, während des Bestehens einer Verbindung auf das Endgerät transportiert werden.

Durch die Replikation von Daten sowie die Anwendung optimistischer Sperrverfahren (weak consistency) wird eine größtmögliche Verfügbarkeit von Daten erreicht. In Folge konkurrierender Zugriffe auf gleiche Datenelemente können bei diesem Verfahren inkonsistente Replikate erzeugt werden. Außerdem kann während der Abkopplung keine Aktualisierung zwischen Replikaten stattfinden. Deshalb sind Mechanismen notwendig, um die Konsistenz zwischen den Replikaten (z. B. nach Beenden einer Abkopplungsphase) wiederherzustellen. Dies erfordert zum einen die Aufzeichnung aller lokal ausgeführten Änderungen während der Abkopplung, sowie die Übermittlung und Ausführung dieser Änderungen auf anderen Replikaten bzw. einem zentralen Server und zum anderen die Erkennung und Behandlung von Konflikten. Abgeleitet aus dem allgemeinen Programmiermodell können abgekoppelte Operationen in die Teilprobleme:

- Vorbereitung auf Abkopplungen,
- Arbeit während der Abkopplung sowie
- Wiederherstellung der Konsistenz und Konfliktbehandlung

zerlegt werden. In Abbildung 2-2 wird das Zustandsmodell des CODA Cache Managers dargestellt, der diese Teilprobleme in separaten Zuständen bzw. durch Verarbeitungsschritte während Zustandsübergängen behandelt.

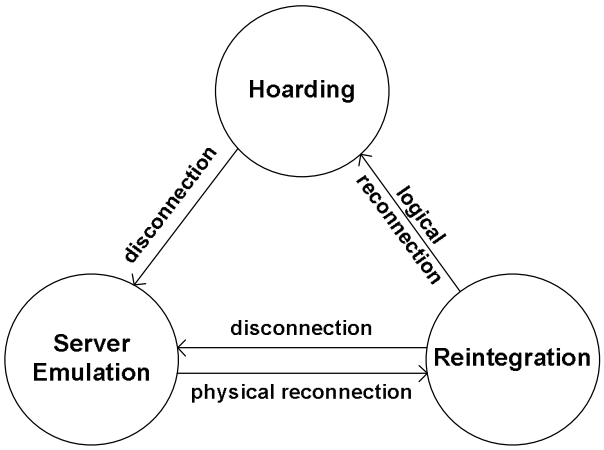


Abbildung 2-2: Zustände des CODA Cache Managers Venus zur Steuerung abgekoppelter Operationen (nach [Sat96])

Es existiert eine Reihe von Ansätzen, die zur Lösung dieser Probleme zum Teil sehr unterschiedliche Mechanismen einsetzen. In verteilten Dateisystemen wie CODA [Sat96, Bra98], Ficus [PGP+90] oder AFS [HuH93] wurden abgekoppelte Operationen auf Dateiebene realisiert. Bayou [DPS+94, TPS+98] ermöglicht die asynchrone Kooperation zwischen potentiell mobilen Rechnern und arbeitet auf der Ebene relationaler Datenbanken. Rover [JLT+95, JoK96, JTK97] unterstützt schmalbandige Verbindungen und Abkopplungen durch die Kombination dynamisch platzierbarer Objekte und asynchroner Aufrufe in Verbindung mit Queueing.

Vorbereitung auf Abkopplungen

CODA besteht aus einer Menge replizierter und verteilter Dateiserver im Festnetz sowie einer Stellvertreterkomponente auf dem Gerät des Benutzers. Diese setzt sich aus dem Cache Manager (Venus) und weiteren Laufzeitkomponenten zusammen [SKK+90]. In Bayou werden ebenfalls nur statische Programmkomponenten eingesetzt. Das System besteht aus einer Menge replizierter relationaler Datenbanken [EMP+97]. Clients können über eine Aufrufschnittstelle auf die Daten zugreifen, eine Komponente im Sinne des Stellvertreters in CODA existiert in Bayou jedoch nicht. Abgekoppelte Operationen werden in diesem System durch leichtgewichtige Datenbankserver erreicht, die auch auf tragbaren Endgeräten installiert werden können. Da Bayou nur eine vollständige Replikation von Datenbanken unterstützt, sind gleichzeitig alle notwendigen Daten auf dem Endgerät verfügbar. Das Vorabladen bzw. Zwischenspeichern von Daten auf dem Client ist deshalb nicht notwendig. In CODA werden dagegen die zuletzt genutzten Dateien im Cache auf dem Client zwischengespeichert. Außerdem kann vom Benutzer eine Liste mit Dateien (hoarding database) definiert werden, die für den Fall einer Abkopplung vorab auf das Gerät geladen werden sollen (prefetching).

Während CODA und Bayou spezifische Lösungen für Datei- bzw. Datenbankzugriffe untersuchen, bietet Rover einen allgemeinen Ansatz. Rover besteht ähnlich CODA aus Programmkomponenten auf Client und Server. Diese stellen ein verteiltes Objektsystem zur Verfügung, mit dem beliebige Anwendungen realisiert werden können. Objekte bestehen aus Programmcode und zugehörigen Daten und können repliziert werden. Die Primärkopie der Objekte befindet sich auf einem Server im Festnetz, Sekundärkopien können von Clients angefordert und dort dynamisch im Cache installiert werden (code shipping). Welche Objekte geladen werden, wird von der jeweiligen Anwendung festgelegt, der Zeitpunkt des Ladens wird vom Laufzeitsystem in Abhängigkeit von der Dienstgüte des Netzwerks gewählt. Alle drei Ansätze verwenden optimistische Sperrverfahren und schwache Konsistenzmodelle für Replikate. In Rover ist zusätzlich das Sperren von Objekten möglich.

Arbeit während der Abkopplung

Wird die Netzwerkverbindung unterbrochen, kann ausschließlich mit den lokal verfügbaren Daten und Funktionen weitergearbeitet werden. Die auf diese Weise ausgeführten Operationen müssen zur späteren Wiederherstellung der Konsistenz aufgezeichnet werden.

In CODA werden Dateiaufrufe vom lokalen Stellvertreter bearbeitet, der die Funktionen des Servers emuliert. Lesezugriffe werden ausgeführt, wenn sich die Datei im Cache befindet, andernfalls wird eine Fehlermeldung zurückgegeben. Schreibzugriffe werden auf der lokalen Kopie der Datei ausgeführt und in einem Protokoll (change modify log) zwischengespeichert. Zur Verringerung der Größe des Protokolls wurden Regeln definiert, mit deren Hilfe sich aufhebende bzw. überschreibende Aufrufe eliminiert werden können [SKM+93].

Die Ausführung von Datenzugriffen wurde in Rover und Bayou auf ähnliche Weise wie in CODA realisiert. In beiden Systemen werden jedoch provisorisch ausgeführte (tentative) und bestätigte (committed) Operationen unterschieden. Schreibzugriffe werden im Zustand der Abkopplung zunächst provisorisch auf dem lokalen Replikat ausgeführt und in einem Protokoll (operation log bzw. write log) aufgezeichnet. Die Änderungen sind für Anwendungen sofort mit dem jeweiligen Status sichtbar und können entsprechend behandelt werden. Zur Optimierung des Aufrufprotokolls können in Rover anwendungsspezifische Prozeduren installiert werden. In Bayou wird die Größe des „write logs“ reduziert, indem

bestätigte Aufrufe durch einen Mechanismus bereits von dem Abgleich mit anderen Replikaten verworfen werden können.

Wiederherstellung der Konsistenz und Konfliktbehandlung

Nach dem Wiederherstellen einer Verbindung muss die Konsistenz zwischen Replikaten und Originaldaten hergestellt werden. Dazu müssen potentielle Konflikte erkannt und aufgelöst werden. In CODA werden die protokollierten Aufrufe nach Wiederankopplung an den Server gesendet, der die Originaldaten zentral verwaltet. Durch den Vergleich der Zeitstempel (storeID) der Serverversion mit der des Replikates können Konflikte erkannt werden. Für Verzeichnisse wurde zusätzlich eine Menge von Regeln festgelegt, die einen Konflikt definieren, da nicht jeder Unterschied zwischen den Zeitstempeln einem Konflikt entspricht [KuS93a, KuS93b]. In Rover werden Konflikte auf ähnliche Weise erkannt [JLT+95]. Die protokollierten Aufrufe werden in diesem System an den Primärserver des jeweiligen Objektes gesendet. Dort wird die mit dem Aufruf übermittelte Version des Sekundärobjektes mit der des Primärobjektes verglichen und bei unterschiedlichen Versionen ein Konflikt erkannt.

Während die Zeitstempel bzw. Versionsvektoren durch das jeweilige System vergeben werden und auf Datei- bzw. Objektebene arbeiten, können in Bayou anwendungsspezifische Konflikte auf Aufrufebene definiert werden [TTP+95]. Jede Schreiboperation enthält dafür eine Datenbankabfrage und das erwartete Ergebnis (dependency check). Stimmt das Ergebnis der Abfrage nicht mit der Erwartung überein, liegt ein Konflikt vor. Im Unterschied zu Zeitstempeln bzw. Versionsvektoren können mit diesem Mechanismus neben Schreib/Schreib- auch Lese/Schreib-Konflikte erkannt werden.

Die Auflösung von Konflikten erfolgt in CODA durch anwendungsspezifische Mechanismen (application specific conflict resolvers) [KuS95]. Diese werden durch Muster einzelnen Dateien bzw. Pfaden zugeordnet und auf dem Client ausgeführt. Die so erzeugte konfliktfreie Version wird als aktuelle Version auf dem Server festgeschrieben. In Rover enthält jede Objektmethode neben der Anwendungsfunktion auch Programmcode zur Auflösung von Konflikten. Die Konfliktauflösung ist in diesem System also Teil des Methodenaufrufes und wird auf dem Home Server des Objektes ausgeführt [JLT+95]. Der Aufruf gilt nach erfolgreicher Abarbeitung auf dem Primärobjekt als bestätigt. In Bayou enthält jede Schreiboperation eine anwendungsspezifische Prozedur zur Konfliktauflösung (mergeproc) [TTP+95]. Diese hat Zugriff auf den gesamten Datenbestand und erzeugt ein alternatives Ergebnis für die Schreiboperation. Nicht-auflösbare Konflikte werden in allen drei Systemen an den Benutzer weitergegeben.

Während in CODA und Rover die Originaldaten auf einem zentralen Server, dem Primärserver, gespeichert werden, arbeiten in Bayou alle Replikate gleichberechtigt und tauschen Schreiboperationen inkrementell über paarweise Verbindungen aus [PST+97]. Die Konsistenz zwischen den einzelnen Replikaten wird erreicht, indem alle Operationen auf allen Replikaten in der gleichen Reihenfolge ausgeführt werden. Mit Hilfe logischer Zeitstempel [Lam78] wird eine globale Ordnung zwischen allen Operationen hergestellt. Bereits ausgeführte (tentative) Operationen müssen deshalb z. T. zurückgerollt werden, wenn ein Schreibzugriff übermittelt wird, der vor diesen Operationen eingeordnet werden muss. Zur Bestätigung von Operationen wird ein Primärserver (primary commit scheme) definiert, der die Reihenfolge für alle Operationen festlegt. Direkt auf dem Primärserver ausgeführte Operationen werden anhand der logischen Zeitstempel geordnet. Operationen, die indirekt, d. h. über andere Replikate zum Primärserver übertragen werden, können nur abhängig vom Zeitpunkt des Eintreffens auf dem Primärserver geordnet und bestätigt werden.

2.1.1.2 Operationen zur Unterstützung schmalbandiger Netzwerkverbindungen

Abgekoppelte Operationen werden durch das Vorabladen und die lokale Speicherung sowie die lokale Verarbeitung von Daten im abgekoppelten Zustand ermöglicht. Sind benötigte Daten oder Verarbeitungsfunktionen lokal nicht verfügbar, können diese während der Abkopplung nicht beschafft werden und verhindern somit in der Regel das Weiterarbeiten. Außerdem sind lokale Änderungen nicht für andere Clients sichtbar, wodurch sich mit der Zeit der Abkopplung auch die Wahrscheinlichkeit von Konflikten erhöht. Nicht zuletzt werden für den Cache und die Aufrufprotokollierung Ressourcen des Clients benötigt, die auf mobilen Endgeräten nur in begrenztem Umfang zur Verfügung stehen.

Abgekoppelte Operationen wurden deshalb um Mechanismen zur adaptiven und dynamischen Nutzung der Ressourcen von Netzwerkverbindungen erweitert, um die Zeiten einer vollständigen Abkopplung zu minimieren. Eine bestehende Verbindung kann zum einen zum Laden neuer Daten bzw. zur Aktualisierung der Daten im Cache, zum anderen zur Vermittlung von lokal aufgezeichneten Änderungen an andere Replikate verwendet werden. Dabei müssen fehlende Daten in Reaktion auf deren Anforderung schnellstmöglich in den Cache geladen werden, um ein möglichst unterbrechungsfreies Weiterarbeiten zu ermöglichen. Für die Aktualisierung des Caches und das Zurückschreiben der Daten ist die Antwortzeit dagegen nicht von Bedeutung. Diese Operationen können unabhängig vom Benutzer zu einem frei wählbaren Zeitpunkt ausgeführt werden. Die Bestimmung des Zeitpunktes hängt von der verfügbaren Verbindung ab. Generell kann durch ein schnelles Zurückschreiben der protokollierten Operationen die Größe des Protokolls reduziert bzw. begrenzt werden. Außerdem wird dadurch die Wahrscheinlichkeit von Konflikten beim Zurückschreiben reduziert. Eine längere Verweilzeit der Operationen im Protokoll ermöglicht dagegen durch die Optimierung des Protokolls (z. B. durch die Eliminierung sich aufhebender Operationen) eine Reduzierung der über das Netzwerk zu übertragenden Daten.

In [HuH95] werden Datenpakete in die drei Klassen interaktive Kommunikation, Lesen von Daten und Zurückschreiben geänderter Daten eingeteilt. Die interaktive Kommunikation erhält die höchste, das Lesen von Daten eine mittlere und das Zurückschreiben die niedrigste Priorität. Für jede der Prioritäten existiert eine Warteschlange. Zur Auswahl der Warteschlange, aus der als nächstes ein Paket versendet werden soll, wird eine Strategie basierend auf Lotterielosen verwendet, die für alle Warteschlangen eine minimale Dienstgüte sichert. Die Anzahl der Lose ist für die Warteschlange unterschiedlich und richtet sich nach deren Priorität, die Auswahl des nächsten Datenpakets erfolgt durch die Ziehung eines Loses.

In einer Erweiterung von CODA [MES95, Sat96] werden implizit ebenfalls zwei Klassen von Paketen unterschieden, indem Operationen zum Zurückschreiben im Hintergrund asynchron an den Dateiserver gesendet werden (trickle reintegration) und Leseoperationen Vorrang vor diesen besitzen. Dabei wird anhand der maximalen Zeit, die ein Benutzer auf eine Datei warten möchte und abhängig von der verfügbaren Datenrate ein Schwellwert für die maximale Größe zu ladender Dateien ermittelt. Für Dateien, die größer als dieser Schwellwert sind, wird eine Fehlermeldung ausgegeben, alle anderen Dateien werden über die Netzwerkverbindung geladen. Die Gültigkeit der zwischengespeicherten Daten wird bei vorhandener Netzwerkverbindung anhand von Zeitstempeln für Dateien und Verzeichnisse (volumes) durch den Client überprüft. Geringe Datenraten werden dabei durch eine Überprüfung mit unterschiedlicher Granularität unterstützt (rapid cache validation). Zunächst werden die Zeitstempel der Verzeichnisse überprüft. Sind deren Zeitstempel noch

gültig, sind auch alle Dateien in diesen Verzeichnissen gültig. Nur wenn Zeitstempel von Verzeichnissen nicht mehr gültig sind, müssen die darin enthaltenen Dateien einzeln geprüft werden. Als verändert erkannte Dateien und Verzeichnisse werden dann im Cache aktualisiert.

Die Reintegration der Daten auf dem Server erfolgt in CODA im Hintergrund und nur dann, wenn das Netzwerk nicht für Leseoperationen benötigt wird. Um die Optimierung des Protokolls der Schreiboperationen zu ermöglichen, wurde für die Einträge eine Mindestverweilzeit definiert. Die Schreiboperationen werden außerdem in Blöcken übertragen. Die Größe eines Blockes wird dynamisch in Abhängigkeit von der verfügbaren Datenrate festgelegt. Überschreitet die Größe einer Schreiboperation die Blockgröße, wird diese in mehrere Fragmente zerlegt, die einzeln übertragen werden. In ähnlicher Weise wird in [HuH95] eine minimale Verweilzeit für protokollierte Schreibzugriffe definiert. Diese hängt von den Eigenschaften der Netzwerkverbindung (z. B. Kosten, Datenrate und Verzögerung) ab und ermöglicht ebenfalls die Reduzierung der zu übertragenden Daten der Schreiboperationen.

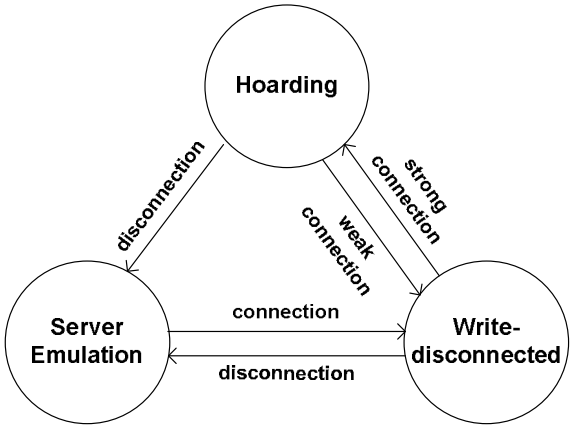


Abbildung 2-3: Zustände des CODA Cache Managers Venus zur Ausnutzung dynamisch wechselnder Netzwerkverbindungen (nach [Sat96])

Die Erweiterung abgekoppelter Operationen hebt die strenge Unterteilung der in [FiG94] beschriebenen Zustände „verbunden“ und „nicht-verbunden“ auf. Abbildung 2-3 stellt die Zustände und Übergange des Cache Managers von CODA als eine Beispiellösung zur Ausnutzung dynamisch wechselnder Netzwerkverbindungen dar. Bei Verfügbarkeit einer Verbindung mit geringer Datenrate arbeitet das System im Zustand „nicht-verbunden“, gleichzeitig werden jedoch Teile der Übergangsfunktionen vom und zum Zustand „verbunden“ ausgeführt. Die Konsistenz der replizierten Daten hängt dabei von der verfügbaren Datenrate sowie weiteren Eigenschaften der Verbindung ab.

2.1.1.3 Verlagerung des Zugriffszeitpunktes

Bei der Unterstützung abgekoppelter Operationen wird als wesentlicher Mechanismus das Vorabladen von Daten eingesetzt. Dabei wird der Zeitpunkt der Anforderung der Daten vor den Zeitpunkt des Zugriffs der Anwendung auf die Daten verlagert [FoZ94]. Zur Realisierung des Vorabladens sind deshalb Vorhersagen für die Daten notwendig, die von der Anwendung in Kürze benötigt werden. Trifft die Vorhersage zu, wird die Antwortzeit für diese Daten wesentlich reduziert, wenn diese bereits lokal vorliegen. Außerdem können auf diese Weise kurzzeitige Verbindungsunterbrechungen und Phasen der Abkopplung überbrückt werden. Beim Abbrechen der Verbindung kann mit den lokal verfügbaren

Daten weitergearbeitet werden. Insbesondere können für das Vorabladen ungenutzte Netzwerkressourcen eingesetzt werden. Falsche Vorhersagen sind dagegen insbesondere bei einer limitierten Datenrate problematisch. Da vorab geladene Daten zusätzlich zu den regulär angeforderten Daten übertragen werden, wird der Kommunikationskanal zusätzlich belastet. Eine wesentliche Voraussetzung für die Effizienz des Vorabladens ist somit ein leistungsstarker Vorhersagemechanismus. Hier können automatische und manuelle Mechanismen unterschieden werden. In CODA wird beispielsweise ein manueller Mechanismus (Hoarding) eingesetzt, bei dem der Benutzer die vorab zu ladenden Dateien vorgibt. Automatische Mechanismen beobachten das Benutzerverhalten und leiten daraus die als nächstes benötigten Daten ab. Beispielsweise können für Web-Seiten die in der aktuell dargestellten Seite enthaltenen Verweise als Hinweise auf die als nächstes aufgerufenen Seiten verwendet werden.

Im Gegensatz zu einem Zugriff auf Daten vor der Benutzung wird der Zugriffszeitpunkt durch Lazy Evaluation und Delayed Write-Back hinter den Zeitpunkt des Zugriffs durch die Anwendung verschoben [FoZ94, Wat94a]. Damit kann die Übertragung vorliegender Daten gleichmäßig über die Zeit verteilt werden, wodurch insbesondere kurzzeitige hohe Belastungen des Netzwerks vermieden werden können. Außerdem können Phasen der Abkopplungen und Verbindungsunterbrechungen überbrückt werden. In [Zen99] wird beispielsweise die Übertragung von E-Mail Nachrichten verzögert, wenn die aktuelle Netzwerklast zu hoch ist. In Rover [JTK97] werden RPC-Nachrichten zunächst durch Queuing zwischengespeichert und erst bei Verfügbarkeit einer Verbindung übertragen. In ähnlicher Weise werden in CODA [Sat96] Schreibzugriffe zunächst lokal ausgeführt und erst an den Primärserver im Festnetz übermittelt, wenn die Bearbeitung der Datei abgeschlossen wurde (Delayed Write-Back). In Wit [Wat94a] werden angeforderte Web-Seiten nur zu dem Teil sofort übertragen, der auf dem Display des Endgerätes dargestellt werden kann. Weitere Daten einer Seite werden verzögert, bis ein Zugriff durch die Anwendung erfolgt (Lazy Evaluation). Außerdem werden Platzhalter für Daten (Futures) anstelle der Originaldaten dargestellt. Der Benutzer kann die Daten durch Verfolgen eines entsprechenden Verweises auf die Originaldaten nachfordern.

2.1.1.4 Autonome Operationen

Während der überwiegende Teil der Systeme abgekoppelte Operationen durch statische Komponenten auf Client und Server unterstützen, erweitert Rover die Möglichkeiten abgekoppelter Operationen durch die Verwendung dynamisch platzierbarer Objekte. Diese bilden autonome Ausführungseinheiten und können nicht nur auf dem Client, sondern auf beliebigen Rechnern platziert und dort unabhängig von einer bestehenden Netzwerkverbindung ausgeführt werden.

Autonome Operationen (siehe [SHZ+99]) stellen eine Verallgemeinerung des Konzeptes der abgekoppelten Operationen dar. Analog zu abgekoppelten Operationen werden diese durch die lokale Verfügbarkeit der zu verarbeitenden Daten sowie der zugehörigen Verarbeitungsfunktionen ermöglicht. Durch die Verwendung mobilen Codes bleiben Operationen jedoch nicht auf einen bestimmten Rechner (speziell das Endgerät) beschränkt, sondern können auf jeden Rechner verlagert werden, der die Verarbeitung mobilen Codes unterstützt. Die lokale Verfügbarkeit von Daten kann in diesem Zusammenhang durch die für abgekoppelte Operationen eingesetzten Mechanismen gesichert werden.

Insbesondere mobile Agenten unterstützen die Ausführung autonomer Operationen auf beliebigen Rechnern innerhalb eines Netzes. Eine Voraussetzung ist jedoch die Verfügbarkeit einer entsprechenden Ausführungsumgebung auf diesen Rechnern. Agenten können

innerhalb dieser Ausführungsumgebung unabhängig vom Benutzer bzw. der Anwendung und damit einer Verbindung zum mobilen Endgerät arbeiten. Insbesondere können sie sich autonom bewegen, um z. B. lokal auf Ressourcen zuzugreifen oder auf Fehlersituationen zu reagieren. Ergebnisse dieser autonomen Verarbeitung können nach Aufbau einer Verbindung zur Anwendung auf dem Endgerät übertragen werden.

2.1.1.5 Ansatz der Anwendungsaufteilung

Durch den Ansatz der Anwendungsaufteilung werden abgekoppelte und autonome Operationen miteinander kombiniert. Das Konzept sieht eine Verteilung der Anwendungs- und Adaptionsfunktionen zwischen Infrastruktur und Endgerät entsprechend der Abhängigkeiten der Funktionen vor. Die Platzierung soll dynamisch entsprechend der aktuellen Ausführungsumgebung angepasst werden. Ziel ist eine optimale Balance zwischen der Ressourcennutzung des Endgerätes und der Abhängigkeit von einer bestehenden Netzwerkverbindung.

In Wit [Wat94a, Wat94b] wird eine Infrastruktur zur Integration von Palmtop Rechnern über drahtlose Netze in ein Festnetz beschrieben. Diese unterstützt die Kommunikation von mobilen Anwendungen über drahtlose Netzwerke mit geringer Bandbreite. Eine wesentliche Rolle spielt dabei das Konzept der Aufteilung und Verteilung von Anwendungsfunktionen zwischen tragbarem Rechner und einem Stellvertreter im Festnetz. Zum einen wird damit das abgekoppelte Arbeiten auf dem tragbaren Rechner, zum anderen die Reduzierung der Datenmenge vor der Übertragung (siehe Tabelle 2-1) realisiert.

Technik	**Realisierung im Proxy bzw. Browser**	**Auswirkung auf die Netzwerkverbindung**
Zwischenspeichern	Caching von häufig benutzten Dokumenten	reduziert Verzögerung und Datenmenge
Vorabladen	Vorabladen der ersten Seite der ersten N Links der aktuellen Seite	verbirgt hohe Antwortzeiten, verteilt burstartige Zugriffe über die Zeit
Datenkompression	Kompression von Dokumenten und Inhalten	reduziert Datenmenge und Verzögerung
Verzögern (Lazy Evaluation)	Anzeige der ersten Seite und Vorbereitung der nächsten Seite (Page Down)	reduziert Datenmenge
Verzögern und Ersetzen (Futures)	Anzeige der ersten Seite und den Rest der Seiten als Future	verbirgt Verzögerung bedingt durch konkurrente Zugriffe, reduziert Datenmenge
Datenreduzierung (Ersetzen)	Anforderung einer ungecachten Seite resultiert in Outline für Seite	reduziert Datenmenge und Verzögerung

Tabelle 2-1: Unterstützungsmechanismen in Wit

Im Gegensatz zu den zuvor beschriebenen Ansätzen arbeiten die Mechanismen in Abhängigkeit der Anwendungsdaten und deren Semantik innerhalb der Anwendung. Beispielsweise wird in einem WWW-Browser für Palm PDAs die Verweisstruktur der Webseiten für das Vorabladen von Seiten ausgenutzt. Außerdem werden die einzelnen Daten zusätzlich mit Informationen über ihre Relevanz für die Gesamtinformation innerhalb einer Seite versehen. Dies erfolgt in Form von Prioritäten für einzelne Datenobjekte, anhand derer eine Filterung der zu übertragenden Daten erfolgt. Im Vergleich zu Ansätzen wie Coda, deren Mechanismen unabhängig von der Semantik der Daten arbeiten, können in Wit Entscheidungen zum Caching, Prefetching sowie zur Filterung abhängig von der Semantik bzw. Relevanz der Daten erfolgen. Der in [Wat95] beschriebene Ansatz zur Strukturierung von Anwendungsdaten in eine Hierarchie von Hyperobjekten, die mit Informationen über die Semantik versehen werden, bleibt nicht auf WWW-Dokumente beschränkt, sondern

kann auf alle Anwendungen übertragen werden, in denen strukturierbare Daten verarbeitet werden (z. B. E-Mail, Video).

2.1.2 Angepasste Kommunikationsprotokolle

Für die Adaption der Datenübertragung wurden auf den verschiedenen Schichten der Übertragungsprotokolle Mechanismen untersucht. Auf den unteren Schichten, vor allem der Sicherungs-, Vermittlungs- und Transportschicht entsprechend des OSI-Modells, können Protokollparameter wie Rahmen- oder Paketgröße, Fenstergrößen zur Flusssteuerung und Mechanismen zur Fehlerbehandlung und Timeouts angepasst werden. Das TCP-Protokoll beinhaltet beispielsweise Mechanismen, mit denen es durch Veränderung der Sendefenstergröße einer Verbindung und Strategien zur Übertragungswiederholung sein Verhalten an Stausituationen und Paketverluste anpassen kann [Ste97]. Diese Mechanismen arbeiten aber unter der Annahme, dass Paketverluste überwiegend aus Überlast- und Stausituationen resultieren. Da dies für drahtlose Verbindungen nicht zutrifft, wurde eine Vielzahl erweiterter Mechanismen zur Leistungssteigerung von TCP über drahtlose Verbindungen untersucht [BPS+97a, XPM+01]. Zur Adaption werden hier das Auftrennen einer Verbindung in zwei separate Verbindungen [BaB95a, BBI+93] sowie angepasste Mechanismen zur Empfangsbestätigung (z. B. selektive Bestätigungen, explizite Verlustmeldungen) [KeM96], zum Zwischenspeichern unbestätigter Pakete und zur Übertragungswiederholung [MMF+96, BSK95] verwendet. Asymmetrische Kommunikationsverbindungen unterliegen vor allem der Problemstellung der Verringerung der nutzbaren Datenrate des Sendekanals durch eine wesentlich geringere Datenrate auf dem Rückkanal. Ziel der Lösungen ist deshalb eine Reduzierung der Datenmenge bzw. der Anzahl von Bestätigungen, die über den Rückkanal gesendet werden. Erweiterte Kommunikationsmechanismen oberhalb der Transportschicht zielen vor allem auf eine Entkopplung zwischen Sender und Empfänger durch eine asynchrone Kommunikation, eine zuverlässige Übertragung durch Queueing-Mechanismen sowie auf Mechanismen zum effizienten Verteilten von Daten und zum Wiederaufsetzen von Verbindungen auf Anwendungsebene.

2.1.2.1 Protokollanpassungen in der Sicherungs- und Transportschicht

Zuverlässige Protokolle der Transportschicht wie z. B. das TCP Protokoll [Pos81] wurden an die Eigenschaften traditioneller drahtgebundener Netzwerke angepasst. Diese zeichnen sich unter anderem durch sehr geringe Bitfehlerwahrscheinlichkeiten aus (z. B. 10^{-12} für Glasfasernetze). Paketverluste treten in diesen Netzen fast ausschließlich in Stausituationen auf, die Verluste durch Übertragungsstörungen betragen laut [Jac88] <<1%. Die Mechanismen zur Fehlerbehandlung basieren auf diesen Eigenschaften. Paketverluste werden dementsprechend z. B. in TCP unter der Annahme behandelt, dass eine Stausituation vorliegt [Jac88, APS99, PaA00]. Einen Überblick über die Staubehandlungsmechanismen in TCP gibt Abbildung 2-4. Der Sender verwaltet zur Steuerung seines Verhaltens zur Staubehandlung einige Variablen. Dies sind die Größen des Sender- (Congestion Window, cwnd) und Empfängerfensters (Receiver Window, rwnd), die maximale Größe eines Segmentes (Sender Maximum Segment Size, SMSS), der Schwellwert zur Unterscheidung zwischen Slow Start und Congestion Avoidance (Slow Start Threshold, ssthresh), die Anzahl der unbestätigt gesendeten Segmente (FlightSize), der Retransmission Timer (RTO), die gemittelte Umlaufzeit (Smoothed Round-Trip Time, SRTT) und die Variation der Umlaufzeit (Round-Trip Time Variation, RTTVAR). G gibt die Granularität der Uhr an, die für die Timer verwendet wird.

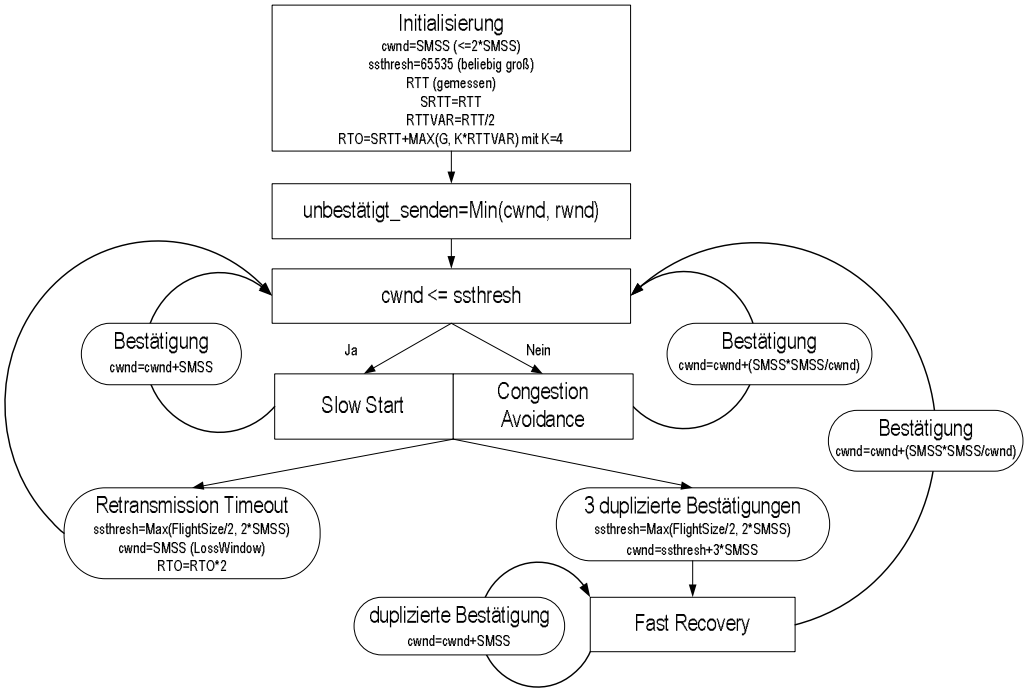


Abbildung 2-4: Überblick über Staubehandlungsmechanismen in TCP

Die Staubehandlung erfolgt mittels der 4 Algorithmen Slow Start, Concestion Avoidance, Fast Retransmit und Fast Recovery, die kombiniert abgearbeitet werden. Demnach werden Paketverluste durch den Ablauf des Retransmission Timers oder durch den Empfang von drei duplizierten Empfangsbestätigungen[1] erkannt. Bei Ablauf des Retransmission Timers wird die Übertragung der verlorenen Daten wiederholt. Außerdem wird die Datenrate des Senders verringert (Congestion Window auf ein Segment zurückgesetzt, cwnd=SMSS) und der Retransmission Timer verdoppelt (RTO = RTO*2). Danach wird die Größe des Congestion Windows entsprechend des Slow-Start Algorithmus' exponentiell erhöht. Nach dem Empfang von drei duplizierten Bestätigungen wird der Verlust des in der Sequenz folgenden Paketes angenommen, auf den mit dem Fast Retransmit und Fast Recovery Mechanismus reagiert wird. Dieser wiederholt die Übertragung des nächsten fehlenden Paketes vor Ablauf des Retransmission Timers. Damit können Überlastsituationen aufgelöst und seltene Paketverluste effizient behandelt werden ohne die Datenrate drastisch zu reduzieren und den Wert des Retransmission Timers zu vergrößern. Danach wird das Congestion Window entsprechend des Congestion Avoidance Algorithmus' linear erhöht. Gebündelte Paketverluste innerhalb eines Sendefensters führen jedoch auch bei Fast Retransmit zu einem Retransmission Timeout und den damit verbundenen Folgen für die Datenrate und den Retransmission Timer [APS99].

Die für zuverlässige Netze getroffenen Annahmen gelten jedoch nicht für drahtlose Netze bzw. Netze mit hoher Fehlerwahrscheinlichkeit. Diese sind im Vergleich zu drahtgebundenen Netzen wesentlich anfälliger gegenüber Störungen des Übertragungsmediums. Außerdem treten infolge von Zellwechseln und Verbindungsunterbrechungen häufig Paketverluste auf, insbesondere auch gebündelt. Wird TCP unverändert für solche Netze verwendet,

[1] In TCP werden gesendete Daten mit einer fortlaufenden Sequenznummer gekennzeichnet. Der Empfänger bestätigt erhaltene Daten gesammelt mit der Sequenznummer der letzten zusammenhängend empfangenen Daten. Werden Pakete mit einer Sequenznummer außerhalb der Reihenfolge empfangen, erhält der Sender für diese Daten jeweils eine Bestätigung für die gleiche Sequenznummer, d. h. duplizierte Bestätigungen.

führt dies zu hohen Leistungsverlusten. Probleme sind vor allem Zellwechsel und häufige Übertragungsfehler. TCP kann Stausituationen nicht von Übertragungsstörungen unterscheiden und wendet in beiden Fällen Mechanismen zur Staubehandlung an. Während Zellwechseln sind mobile Geräte kurzzeitig nicht erreichbar und gesendete Pakete können nicht zugestellt werden. Der Ablauf des Retransmission Timers signalisiert den Paketverlust, wodurch dieser verdoppelt und der Slow-Start Mechanismus aktiviert wird. Aufeinanderfolgende Paketverluste resultieren demnach in einem exponentiell steigenden Wert für den Retransmission Timer und einer langsam wachsenden Datenrate. Dadurch wird lange auf ein erneutes Ablaufen des Zählers gewartet und mit geringer Datenrate erneut gesendet, anstatt die nicht übermittelten Pakete möglichst schnell erneut zu versenden und die Übertragungsgeschwindigkeit beizubehalten. Ähnliches gilt für Übertragungsstörungen. Auch hier führt die Behandlung der Paketverluste als Stausituation zu einer schlechten Auslastung der ohnehin geringen Übertragungskapazität und damit zu Leistungseinbußen und hohen Antwortzeiten [CaI95].

Zur Lösung dieser Problemstellung können nach [BPS+97a] zwei grundlegende Ansätze unterschieden werden:

1. Das Verbergen von Paketverlusten, die nicht aus Stausituationen resultieren, vor dem Sender und
2. die differenzierte Behandlung von Paketverlusten, die durch Staus bzw. Zellwechsel oder Übertragungsstörungen verursacht wurden.

Eine Übersicht über weitere Unterklassen der beiden generellen Ansätze enthält Abbildung 2-5. Die einzelnen Ansätze werden nachfolgend beschrieben.

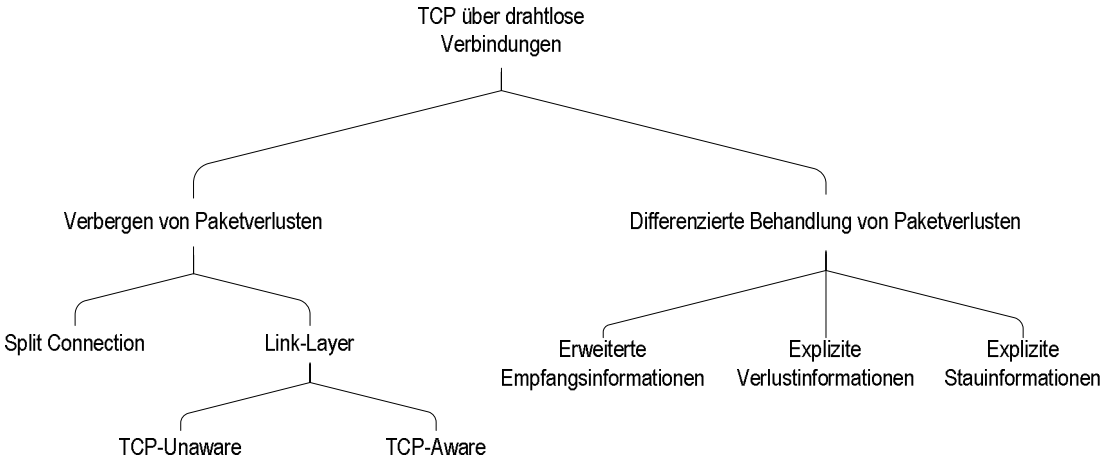


Abbildung 2-5: Ansätze zur Leistungssteigerung von TCP über drahtlose Netzwerke

Verbergen von Paketverlusten

Durch das Verbergen von Paketverlusten vor TCP soll vor allem eine Leistungssteigerung der Datenübertragungen erreicht werden ohne das Transportprotokoll selbst zu ändern.

Split Connection-Ansatz

Erste Ansätze zur Verbesserung der Performance von TCP über drahtlose Netzwerke schlugen nach dem so genannten Split-Connection-Verfahren die Auftrennung heteroge-

ner[2] Verbindungen in homogene Teilverbindungen vor [BBI+93, BaB95a]. Damit wird für jede Teilverbindung eine separate Flusssteuerung und Staubehandlung durchgeführt. Paketverluste durch Übertragungsstörungen oder Staus einer Teilverbindung können so vor anderen Teilverbindungen verborgen werden. Typisch ist die Auftrennung der Verbindung auf der Basisstation, dem Verbindungsrechner zwischen drahtlosem und drahtgebundenem Netzwerk. Dazu wird eine Softwarekomponente auf der Basisstation installiert, die die beiden Teilverbindungen verwaltet und gegebenenfalls entsprechend des Proxy-Ansatzes zusätzliche Funktionen auf verschiedenen Protokollschichten realisieren kann (z. B. Session Management oder Verschlüsselung) [BBI+93].

Außerdem wird der Einsatz unterschiedlicher Protokolle auf den einzelnen Teilverbindungen, angepasst an das jeweilige Übertragungsmedium, möglich. Während in [BaB95a] für beide Teilverbindungen das unveränderte TCP-Protokoll eingesetzt wird, werden in [Haa97], [WaT98] und [YaB94] spezielle Protokolle verwendet. Unter der Annahme einer Single-Hop-Verbindung zwischen mobilem Rechner und der Basisstation werden in diesen Protokollen vereinfachte Verfahren zur Fehlerkorrektur und Flusskontrolle eingesetzt. Durch verkürzte Verbindungskennungen und Sequenznummern kann außerdem ein verkleinerter Nachrichtenkopf verwendet werden, um die zu übertragende Datenmenge zu reduzieren. In [Haa97] und [WaT98] wird insbesondere ein asymmetrischer Ansatz verfolgt, nach dem die Komplexität der Protokollverarbeitung auf die Basisstation verlagert wird. Auf dem mobilen Endgerät wird dadurch weniger Rechenleistung und Energie benötigt. In [YaB94] werden die Pakete auf der Basisstation zerlegt bzw. zusammengesetzt, um die Paketgröße an die maximale Übertragungseinheit (Maximum Transmission Unit, MTU) der jeweiligen Teilverbindung anzupassen. Die Auftrennung der Verbindung ist generell ohne Änderungen des TCP-Protokolls möglich. Werden über die drahtlose Verbindung spezielle oder angepasste Protokolle eingesetzt, bleibt dies dennoch für stationäre Rechner im Internet transparent.

Durch die Auftrennung der Verbindung wird jedoch die Ende-zu-Ende Semantik von TCP verletzt, d. h. bestätigte Daten müssen nicht notwendigerweise vom Empfänger erhalten worden sein[3]. Weiterhin wird die Ende-zu-Ende Verzögerung von Verbindungen erhöht, da jedes Paket die Protokollverarbeitung von TCP doppelt durchlaufen muss. Insbesondere wird auch auf der Basisstation eine Verarbeitung zur Verwaltung der Teilverbindungen notwendig. Dementsprechend muss eine zusätzliche Softwarekomponente auf allen Basisstationen installiert werden. Außerdem verwaltet die Komponente auf der Basisstation den Zustand der Teilverbindungen, der bei einem Zellwechsel an die neue Basisstation übermittelt werden muss. Der Ansatz erfordert damit für jede Verbindung zusätzliche Ressourcen auf der Basisstation und führt zu einem erhöhten Aufwand bei Zellwechseln.

Link-Layer Protokolle

Eine weitere Möglichkeit, Paketverluste vor der Transportschicht zu verbergen, ist die Fehlersicherung auf Schichten unterhalb der Transportschicht, i. A. der Sicherungsschicht. Damit kann für höhere Schichten eine zuverlässige Übertragung angeboten werden ohne dass diese verändert werden müssen. Für TCP besteht demnach nicht mehr das Problem der Unterscheidung zwischen Stau und Übertragungsstörungen. Paketverluste können

[2] Die Heterogenität bezieht sich auf die Leistungsmerkmale von Netzwerktechnologien (z. B. Datenrate, Verzögerung, Fehlerwahrscheinlichkeit).

[3] Eine Ausnahme bildet hier Mobile-TCP, bei diesem Ansatz werden die Bestätigungen auf der Basisstation zurückgehalten, bis die Bestätigung der anderen Teilverbindung eintrifft.

wieder eindeutig als Überlastsituationen interpretiert und als solche behandelt werden. Außerdem wird im Gegensatz zu den Split-Connection Verfahren die Ende-zu-Ende Semantik der TCP-Bestätigungen bewahrt.

Fehlersicherungsverfahren können in Fehlerkorrektur- und Fehlerbehandlungsverfahren unterschieden werden. Bei fehlerkorrigierenden Verfahren spricht man auch von Vorwärtsfehlerkorrektur (Forward Error Correction, FEC). Dabei fügt der Sender einem Datenwort so viel Redundanz hinzu, dass der Empfänger eine bestimmte Anzahl von Fehlern korrigieren kann. Bei fehlerbehandelnden Verfahren werden Datenworte durch ein Verfahren zur Fehlererkennung auf Korrektheit geprüft. Wird ein Datenwort als nicht korrekt erkannt, wird es durch den Empfänger erneut beim Sender angefordert (Automatic Repeat Request, ARQ). Während durch das Hinzufügen von Redundanz bei Vorwärtsfehlerkorrektur ein erhöhter Aufwand bei der Datenübertragung entsteht, setzen Verfahren zur Fehlerbehandlung einen Rückkanal zwischen Sender und Empfänger voraus. Kombinationen beider Verfahren sind möglich und zielen vor allem auf ein optimales Verhältnis von Redundanz und Übertragungswiederholungen.

Generell können Link-Layer Protokolle unterschieden werden, die unabhängig von Protokollen anderer Schichten arbeiten (TCP-Unaware) oder Informationen über Protokolle anderer Schichten, insbesondere der Transportschicht einbeziehen (TCP-Aware). Airmail [APL+95] ist ein Protokoll, das Fehlersicherungsverfahren unabhängig von TCP einsetzt. Dabei werden Vorwärtsfehlerkorrektur und Übertragungswiederholung in verschiedenen Stufen miteinander kombiniert, um den Aufwand für die Fehlersicherung an die aktuelle Verbindungsqualität anzupassen. Das Protokoll ist ebenfalls asymmetrisch, verlagert also die Komplexität der Funktionen auf die Basisstation. Weitere Beispiele für unabhängig arbeitende Sicherungsschichten sind die entsprechenden Schichten in Bluetooth, WLAN und IrDA (siehe [Sch03]).

Bei diesen Verfahren werden parallel sowohl auf der Sicherungsschicht als auch auf der Transportschicht Mechanismen zur Fehlersicherung eingesetzt. Aus der Unabhängigkeit beider Schichten kann eine gegenseitige Beeinflussung resultieren, die eine Reduzierung der Leistung der Datenübertragung zur Folge hat. So können durch nicht abgestimmte Timeouts Übertragungswiederholungen sowohl auf der Sicherungs- als auch auf der Transportschicht ausgelöst werden. Außerdem spielt auf der Sicherungsschicht die Reihenfolge der Pakete keine Rolle. Durch Änderung der Reihenfolge können aber auf der Transportschicht duplizierte Bestätigungen versendet und damit Übertragungswiederholungen durch den Fast-Retransmit-Mechanismus ausgelöst werden. Als Folge werden Übertragungswiederholungen doppelt durchgeführt bzw. Staubehandlungsmechanismen ausgelöst. Diese Beeinflussung wurde z. B. in [WoL99] untersucht. Demnach resultieren aus einfachen Fehlerkorrekturmechanismen und hoher Datenrate (WLAN) erhebliche Leistungseinbußen, die aber durch Verwendung effizienter Fehlerkorrekturmechanismen über drahtlose Verbindungen mit geringen Datenraten (CDMA) wesentlich geringer sind.

Ansätze, die mit Informationen über die Transportschicht arbeiten, zielen deshalb auf eine Vermeidung der Beeinflussung der Fehlersicherungsmechanismen auf beiden Schichten. Das bekannteste Beispiel ist das Snoop-Protokoll (siehe [BSK95], [BSA+95] und [BPS+97a]). Dieses basiert auf einer zusätzlichen Komponente, dem Snoop-Agenten, auf der Basisstation. Der Snoop-Agent wird in den Routingcode der Basisstation eingefügt. Dadurch können alle TCP-Pakete und Bestätigungen überwacht werden, ohne den Paketfluss zu ändern. Der Snoop-Agent speichert alle Pakete, die an mobile Rechner versendet und noch nicht bestätigt wurden, in einem Zwischenspeicher. Paketverluste werden durch duplizierte Bestätigungen vom mobilen Rechner oder durch das Ablaufen lokaler Zähler

erkannt. Daraufhin wiederholt der Snoop-Agent die verlorenen Pakete mittels der lokal zwischengespeicherten Daten. Außerdem werden duplizierte Bestätigungen nicht an den Sender weitervermittelt, um das Auslösen von Fast-Retransmit zu verhindern.

WTCP [RaM98] verfolgt einen ähnlichen Ansatz. Auch hier wird eine Komponente in den Routingcode der Basisstation eingefügt, die Pakete zum mobilen Rechner zwischenspeichert und lokale Übertragungswiederholungen durchführt. Außerdem werden bei WTCP die Round-Trip-Zeiten der Pakete korrigiert, um die Berechnung des Retransmission Timers beim Sender nicht zu beeinflussen. Beide Ansätze unterstützen nur die Übertragung mit mobilen Rechnern als Empfänger. Für Übertragungen von mobilen zu stationären Rechnern sieht der Snoop-Ansatz negative Bestätigungen zur Leistungsverbesserung vor. Auf der Basisstation werden verlorene Pakete anhand von Unterbrechungen innerhalb einer Paketfolge oder einer definierten Zeitspanne ohne Paketempfang erkannt. Für diese Pakete erzeugt die Basisstation negative Bestätigungen, die mit den normalen Bestätigungen übermittelt werden. Mittels dieser kann der mobile Rechner die verlorenen Pakete selektiv wiederholen. Dies setzt aber voraus, dass der mobile Rechner das Verfahren negativer Bestätigungen unterstützt.

Verfahren, die mit Kenntnis über das Transportprotokoll arbeiten, vermeiden also eine Überlagerung der Fehlersicherungsmechanismen der Sicherungs- und Transportschicht. Auf der Basisstation wird eine zusätzliche Komponente eingefügt. Deren Zustand kann, muss aber bei einem Handover nicht übertragen werden. Handover werden damit nicht aufwendiger. Das Fehlen der Pakete im Zwischenspeicher auf der neuen Basisstation hat nur eine kurzzeitige Leistungseinbuße zur Folge. Das TCP-Protokoll kann ohne Änderungen eingesetzt werden. Ein wesentlicher Nachteil ist, dass durch die Auswertung von Protokollköpfen der Transportschicht (TCP) auf der Vermittlungsschicht (IP) die Transparenz zwischen den Schichten aufgehoben wird. Einzelne Schichten können somit nicht mehr einfach ausgetauscht werden. Ein weiteres Problem ergibt sich aus der Verschlüsselung von TCP-Paketen. Wird auch der Nachrichtenkopf verschlüsselt, sind diese Informationen für Snoop und ähnliche Protokolle nicht mehr zugänglich. Die Lösung durch Entschlüsselung der Pakete auf den möglicherweise nicht vertrauenswürdigen Basisstationen birgt erhebliche Sicherheitsrisiken.

Differenzierte Behandlung von Paketverlusten

Die nachfolgend vorgestellten Lösungen passen das TCP-Protokoll durch erweiterte Mechanismen an drahtlose Verbindungen an. Es werden also, anders als bei den zuvor beschriebenen Lösungen, Änderungen des TCP-Protokolls notwendig. Die Grundidee ist es, den Kommunikationspartnern Informationen über wichtige Ereignisse während der Datenübertragung zugänglich zu machen, damit diese entsprechend darauf reagieren können. Zum einen können erweitere Informationen über die beim Empfänger eingetroffenen Daten an den Sender übermittelt werden. Zum anderen wird der Sender explizit über die Verlustursache von Paketen informiert und kann dann entsprechende Mechanismen aktivieren.

Erweiterte Empfangsinformationen

Paketverluste werden in TCP durch Retransmission Timeouts oder den Empfang duplizierter Bestätigungen erkannt. Beim Empfang dreier duplizierter Bestätigungen wird in TCP beim Sender der Fast Retransmit und Recovery Mechanismus ausgelöst. Dieser wiederholt das als nächstes vom Empfänger erwartete Segment unter der Annahme, dass dieses Segment verloren wurde. Da die darauf folgenden Segmente aber beim Empfänger eintrafen und die duplizierten Bestätigungen auslösten, können weitere Pakete übertragen werden.

Aus diesem Grund wird auch der Slow-Start-Mechanismus nicht ausgelöst. Eine weitere Verbesserung der Fehlerbehandlung kann durch zusätzliche Informationen über beim Empfänger eingetroffene Segmente in den Bestätigungen erreicht werden. Dieser Ansatz wird unter anderem mit der Einführung selektiver Bestätigungen (Selective Acknowledgements, SACK) [MMF+96] und dem SMART Retransmission Mechanismus [KeM96] verfolgt. Beide Lösungen verwenden Bestätigungen, die neben der gewöhnlichen TCP-Bestätigung weitere Informationen über Segmente enthalten, die außerhalb der Reihenfolge empfangen wurden. Selektive Bestätigungen (SACKs) können bis zu drei kumulierte Bereiche von Sequenznummern enthalten, Bestätigungen nach [KeM96] enthalten zusätzlich die Sequenznummer des Segmentes, das die duplizierte Bestätigung auslöste. Anhand dieser Informationen kann der Sender verlorene Segmente identifizieren und diese erneut übertragen, ohne auf den Ablauf des Retransmission Timers zu warten. Damit können gehäufte Paketverluste innerhalb eines Sendefensters effizient behandelt werden. Paketverluste werden aber entsprechend des Fast Retransmit und Recovery-Mechanismus immer noch als Stausituationen behandelt.

Explizite Verlustinformationen

Eine differenzierte Behandlung von Stausituationen und Übertragungsstörungen sollen Ansätze ermöglichen, die explizite Informationen über die aktuelle Netzwerksituation (z. B. Füllstand von Warteschlangen) ermitteln und die Ursachen für Paketverluste aus diesen Informationen ableiten und an den Sender übermitteln. Explizite Verlustmeldungen (Explicit Loss Notification, ELN) können auf unterschiedliche Weise implementiert werden. Eine Realisierungsmöglichkeit wurde im Rahmen des Snoop-Protokolls [BaK98] beschrieben. Bei diesem Ansatz wird der Datenaustausch von mobilen Rechnern zu stationären Rechnern unterstützt. Der Snoop-Agent auf der Basisstation hört dabei den Datenverkehr ab und speichert die Sequenznummern der empfangenen Daten. Unterbrechungen in der Folge der Sequenznummern kennzeichnen Paketverluste. Zur Unterscheidung von Verlusten durch Stau oder Störungen enthält jedes vom mobilen Rechner gesendete Segment die Länge der Warteschlange auf dem mobilen Rechner. Der Snoop-Agent kann außerdem die Länge der Warteschlange der Basisstation ermitteln. Empfängt der Snoop-Agent ein Segment außerhalb der Reihenfolge, werden die Längen beider Warteschlangen mit Schwellwerten verglichen. Überschreitet eine der Längen den entsprechenden Schwellwert, wird daraus eine Stausituation abgeleitet, andernfalls eine Störung. Bestätigungen, die mit aufgezeichneten Unterbrechungen korrespondieren, werden durch ein ELN-Bit im Nachrichtenkopf entsprechend als Verlust durch Stau oder Störung gekennzeichnet. Bei gesetztem ELN-Bit wiederholt der Sender das entsprechende Segment, ohne weitere Mechanismen zur Staubehandlung auszulösen.

In [DiJ01a, DiJ01b] soll die Datenübertragung vom stationären zum mobilen Rechner durch explizite Verlustinformationen unterstützt werden. Bei dieser Lösung wird ähnlich dem Snoop-Ansatz eine Stellvertreterkomponente auf der Basisstation installiert. Diese Komponente speichert die außerhalb der Reihenfolge erhaltenen Segmente und bewertet die Ursache von Paketverlusten. Zur Vermittlung der Informationen über Segmentverluste und deren Ursache wird eine neue Form von Bestätigungen verwendet. Diese enthält die Sequenznummern verlorener Segmente und ein ELN-Bit zur Kennzeichnung der Verlustursache. Die Bestätigungen werden vom mobilen Rechner erzeugt und an die Basisstation gesendet, wenn Unterbrechungen in der Segmentfolge erkannt werden. Auf der Basisstation wird anhand der Sequenznummern im Cache nach den entsprechenden Segmenten gesucht. Befindet sich ein Segment im Cache, wurde es erst auf der drahtlosen Verbindung verloren, andernfalls ist die Ursache des Verlustes ein Stau im Festnetz. Die Komponente

auf der Basisstation setzt das ELN-Bit entsprechend und vermittelt die Bestätigung weiter an den Sender. Dieser erhält mit der Bestätigung eine explizite Information über Paketverluste und deren Ursache und kann entsprechende Mechanismen zur Fehlerbehandlung aktivieren.

In [Man03] soll ebenfalls die Übertragung von stationären zu mobilen Rechnern unterstützt werden. Die Ermittlung der Verlustursache erfolgt bei dieser Lösung jedoch auf dem Senderrechner. Der Sender schätzt dazu die aktuelle Belastung des Netzwerkes. Beim Erkennen eines Paketverlustes identifiziert er dessen Ursache durch Anwendung einer Heuristik auf den Netzwerkzustand und reagiert entsprechend. Bei Paketverlusten durch Störung wird ein adaptiver Mechanismus zur Übertragungswiederholung eingesetzt. Demnach wird für die Übertragungswiederholung eine kleinere Segmentgröße verwendet. Dies reduziert die Paketfehlerrate und die Datenmenge, die bei einem erneuten Verlust nochmals wiederholt werden muss. Außerdem werden durch kleinere Segmente bei der gleichen Datenmenge mehr Bestätigungen erzeugt, wodurch die Aktivierung des Fast Retransmit Mechanismus bei Paketverlusten wahrscheinlicher wird.

In [CaI95] wird eine Lösung zur expliziten Benachrichtigung der Kommunikationspartner bei Zellwechseln beschrieben. Die TCP-Implementierung auf dem mobilen Rechner wird zunächst von der darunter liegenden Schicht über den Abschluss des Zellwechsels benachrichtigt und ruft den Mechanismus Fast Retransmit auf. Zur Benachrichtigung des stationären Rechners erzeugt der mobile Rechner drei duplizierte Benachrichtigungen und sendet diese an den stationären Rechner. Auf diesem wird nach Erhalt der Benachrichtigungen ebenfalls der Fast Retransmit Mechanismus aktiviert. Damit konnte eine signifikante Verbesserung der Verzögerung sowie der Übertragungsrate bei Zellwechseln erreicht werden. Weitere Probleme der Datenübertragung über drahtlose Verbindungen wurden in [CaI95] jedoch nicht adressiert.

Explizite Stauinformationen

Eine Benachrichtigung über Stausituationen wird in [RFB01] beschrieben. Router markieren dazu Datenpakete, die einer Verzögerung unterliegen, mit einer Explicit Congestion Notification (ECN). Die markierten Pakete erreichen zunächst den Empfänger, der die Staubenachrichtigung in Form eines gesetzten ECN-Echo-Flags an den Sender weitergibt. Der Sender kann dann mit dem Absenken der Senderate reagieren. Durch die explizite Benachrichtigung kann der Sender zwischen Stau und Verlusten unterscheiden. Zur Realisierung des Mechanismus sind jedoch Änderungen sowohl beim Sender und Empfänger als auch in den Routern notwendig, wodurch eine Einführung erschwert wird.

2.1.2.2 Anpassung an asymmetrische Verbindungen

Einen besonderen Problembereich für TCP/IP stellt die Übertragung über asymmetrische Kommunikationskanäle dar. Asymmetrie kann dabei hinsichtlich der Datenrate, des Medienzugriffs oder der Fehlerrate bestehen [BaP01]. Die größte Bedeutung hat dabei die Asymmetrie der Datenrate, die in der Praxis häufig vorzufinden ist. Bedingt ist diese durch asymmetrische Kommunikationskanäle wie sie z. B. Bluetooth oder DECT (siehe [Sch03]) aber auch drahtgebundene Technologien wie ADSL [Füh00] anbieten oder durch unidirektionale Verteiltechnologien wie DAB [ETS00] und DVB [Rei00], bei denen ein interaktiver Rückkanal über eine meist schmalbandigere Verbindung (z. B. über Modem bzw. ein flächendeckendes Funknetz wie GSM bzw. UMTS [Sch03]) ergänzt wird. In beiden Fällen ist aus Sicht des Benutzers die Datenrate zum Empfangen (Downstream) von Daten wesentlich größer als zum Senden von Daten (Upstream). Dies ist durch technische oder

ökonomische Gegebenheiten aber auch durch die Eigenschaften des Web-Zugriffs motiviert, bei dem wesentlich mehr Daten zum Benutzer übertragen werden, als in der Gegenrichtung.

Hauptproblem einer wesentlich geringeren Datenrate des Rückkanals (Upstream) ist die Verringerung der erreichbaren Datenrate auf dem Sendekanal. Dies resultiert aus dem zeitlichen Zusammenhang zwischen Datenpaketen und deren Bestätigungen bei TCP/IP. Der Sender sendet zunächst entsprechend der Größe des Sendefensters eine bestimmte Anzahl von Datenpaketen ohne Bestätigung. Erst nach dem Eintreffen von Bestätigungen können dann weitere Daten gesendet werden. Eine zu geringe Datenrate des Rückkanals führt zu einer Sammlung von Bestätigungen in den Warteschlangen der Router und damit zu Verzögerungen der Bestätigungen. Aufgrund der begrenzten Länge der Warteschlangen in Routern werden Bestätigungen zur Stauvermeidung verworfen, wenn die Warteschlangenlänge einen bestimmten Schwellwert überschreitet. Dies wird beim Sender als Verlust der gesendeten Daten interpretiert. Infolge der Staubehandlung wird daraufhin die Senderate verringert. Erfolgt keine Unterstützung der Asymmetrie, resultiert daraus ein hoher Verlust von Sendekapazität [BPS+97b].

Eine Lösungsmöglichkeit für diese Problemstellung ist die Reduzierung der Datenmenge der Bestätigungen. Dies kann durch die Reduzierung der Größe der einzelnen Bestätigungen oder durch die Verringerung der Anzahl der Bestätigungen erfolgen.

Kompression der Paketköpfe von Bestätigungen

Die Verkleinerung einzelner Bestätigungen kann durch die Kompression der Paketköpfe erreicht werden [Jac90]. Bei diesem Verfahren wird ausgenutzt, dass ein Teil der Daten im Paketkopf über die gesamte Verbindung konstant bleibt. Diese werden einmalig während des Verbindungsaufbaus übertragen. Alle weiteren Paketköpfe bestehen dann nur noch aus den veränderlichen Daten und einer Verbindungskennung. Die vollständigen Paketköpfe werden beim Empfänger durch die vorliegenden Verbindungsdaten rekonstruiert. Durch die Komprimierung der Paketköpfe kann eine Reduzierung der Größe von Bestätigungen um mehr als 50% erreicht werden.

Acknowledgement Filtering

Verfahren zur Reduzierung der Anzahl der Bestätigungen sind Acknowledgement Filtering (AF) und Acknowledgement Congestion Control (ACC) (siehe [BPS+97b]). Acknowledgement Filtering erweitert die Verarbeitung in Routern und nutzt die Eigenschaft der kumulierten Bestätigungen in TCP. Für eintreffende Bestätigungen wird in der Warteschlange des Routers nach Bestätigungen gesucht, die in der aktuellen Bestätigung enthalten sind. Diese damit redundanten Bestätigungen werden verworfen, wodurch die Datenmenge über den Rückkanal reduziert wird. Damit werden aber auch duplizierte Bestätigungen verworfen, wodurch der Fast Recovery Mechanismus beeinflusst wird.

Acknowledgement Congestion Control

Acknowledgement Congestion Control modifiziert dagegen die TCP-Implementierung des Empfängers, um die Anzahl der Bestätigungen an die Datenrate des Rückkanals anzupassen. Dazu wird der Explicit Congestion Notification (ECN) Mechanismus [RFB01] auch für Bestätigungen angewendet. Grundlage ist die Markierung von Bestätigungen in Routern, deren Warteschlange einen bestimmten Schwellwert überschritten hat. Diese Markierung weist auf einen Stau bei der Übertragung der Bestätigungen hin. Die Markierung der

Bestätigungen wird beim Sender auf die Datenpakete übertragen und erreicht so den Empfänger. Dieser passt daraufhin die Häufigkeit für das Senden von Bestätigungen an, d. h. es wird erst für eine größere Menge von empfangenen Daten eine Bestätigung gesendet. Die Häufigkeit der Bestätigungen wird wieder erhöht, wenn die Datenpakete keine Markierung mehr enthalten.

Vergleich der Verfahren

Im Vergleich zum unveränderten TCP Protokoll kann mit der Kompression der Paketköpfe die größte Leistungssteigerung erzielt werden. Durch die Halbierung der Größe von Bestätigungen kann nahezu eine Verdopplung der Sendekapazität beobachtet werden. Im Vergleich zwischen ACC und AF arbeitet das AF-Verfahren effektiver. Dieses stellt sicher, das sich pro Verbindung nur eine Bestätigung in der Warteschlange eines Routers befindet. Das ACC-Verfahren reduziert die Anzahl der Bestätigungen nur in geringerem Maße. Eine Kombination der Verfahren ist nur bei sehr großen Differenzen zwischen Sende- und Bestätigungskanal sinnvoll. Entsprechend der Untersuchungen in [BPS+97b] mit einem Sendekanal mit 10 Mbit/s konnten bei einem Rückkanal mit mindestens 28,8 Kbit/s beim Einsatz der Kompression der Paketköpfe durch ACC oder AF keine weitere Steigerung der Sendekapazität mehr erreicht werden.

2.1.2.3 Erweiterte Kommunikationsmechanismen oberhalb der Transportschicht

RPC-Erweiterungen

Der Remote Procedure Call (RPC) ist ein wichtiges Paradigma für die Implementierung von Client/Server-Systemen. Der RPC wird auch heute noch als Realisierungsbasis verteilter Systeme eingesetzt (z. B. in Form von Java RMI [Sun02] und in CORBA [OMG02b]). Der RPC stellt als synchroner Aufrufmechanismus hohe Anforderungen an die Verfügbarkeit einer Kommunikationsverbindung zwischen Client und Server. Das Binden zwischen Client und Server erfolgt z. B. in Java RMI vor dem ersten Aufruf zur Laufzeit. Serverausfälle bzw. Serverwechsel müssen explizit unterstützt werden. Außerdem werden Aufrufe überwiegend entsprechend des Programmablaufes abgesetzt. Insbesondere erfolgt keine Optimierung der Zahl und Zeitpunkte der Aufrufe. Diese Anforderungen werden in mobilen Infrastrukturen durch die hohen Fehlerraten drahtloser Netzwerke bzw. die Mobilität der Endgeräte nicht erfüllt.

In [BaB95b] wird deshalb der RPC durch verschiedene Mechanismen zu einem M-RPC mit Mobilitätsunterstützung erweitert. Die Erweiterungen umfassen das dynamische Binden zwischen Client und Server, eine zuverlässige Aufrufvermittlung über unzuverlässige Verbindungen und eine Unterstützung abgekoppelter Operationen. Der Ansatz basiert auf einer Stellvertreterkomponente, die auf der Basisstation im Festnetz installiert wird. Ähnlich I-TCP [BaB95a] wird die Verbindung zwischen Client und Server in zwei Teilverbindungen zerlegt, auf der unterschiedliche Transportprotokolle verwendet werden. Um die hohe Fehlerrate drahtloser Verbindungen zu kompensieren, verwendet M-RPC ein zuverlässiges Transportprotokoll (Reliable Data Protocol, RDP) zwischen mobilem Rechner und Basisstation. Im Festnetz werden RPC-Nachrichten über UDP bzw. TCP übertragen. Nachrichtenverluste im Festnetz und Serverausfälle werden durch die Komponente auf der Basisstation behandelt. Diese speichert RPC-Requests, die von einem Client korrekt empfangen wurden in einem Cache, solange für diese noch keine entsprechende Antwort vor-

liegt. Ähnlich dem Snoop-Protokoll [BPS+97a] werden die Requests von der Komponente auf der Basisstation wiederholt, um eine erneute Übertragung über die drahtlose Verbindung zu vermeiden. In ähnlicher Weise werden Verbindungsunterbrechungen durch den Stellvertreter unterstützt. Dieser speichert ankommende RPC-Nachrichten in einer Warteschlange, bis der mobile Rechner wieder erreichbar ist. Durch eine indirekte Bindung des Clients an den Server über den Stellvertreter kann eine dynamische Bindung zur Laufzeit erfolgen. Der Client erhält vom Stellvertreter eine logische Bindung, über die er alle RPC-Nachrichten absetzt. Der Stellvertreter bindet sich transparent für den Client an einen Dienst und vermittelt die Client-Aufrufe an diesen. Bei Serverausfällen und nach einem Handover kann der Stellvertreter eine neue Bindung zu einem anderen Server herstellen, ohne dass der Client diese Veränderung bemerkt. Ein Neubinden ist dabei zu jedem Zeitpunkt nur bei zustandslosen Servern möglich.

Eine wesentliche Erweiterung des RPC stellen asynchrone Aufrufe dar. Diese werden durch Futures [WFN90] bzw. Promises [LiS88] ermöglicht, die als Platzhalter für die Ergebnisse von RPC-Aufrufen dienen. Von Anwendungen abgesetzte RPC-Aufrufe kehren nach dem Versenden des Aufrufs mit einem Future bzw. Promise als Resultat des Aufrufs zurück. Die Anwendung kann dann weiterarbeiten und auch weitere RPC-Aufrufe absetzen. Über die Platzhalterobjekte steht der Zustand bzw. das Ergebnis eines Aufrufs zur Verfügung. Der Aufrufer wird also nicht blockiert, bis das Ergebnis eines Aufrufs eintrifft.

In Rover [JoK96] wird der RPC-Mechanismus ebenfalls erweitert, um entfernte Aufrufe asynchron auszuführen. Außerdem werden Mechanismen bereitgestellt, um Verbindungsabbrüche und Abkopplungen zu behandeln. Ruft eine Anwendung einen RPC auf, wird der Aufruf zur weiteren Verarbeitung im Operation Log persistent gespeichert (Queueing). Danach kehrt der Aufruf zurück und die Anwendung kann weiterarbeiten. Als Resultat des Aufrufes erhält der Aufrufer ein Promise-Objekt [LiS88] zur späteren Übergabe bzw. Abfrage der Aufrufergebnisse. Registriert der Aufrufer einen Callback, wird er beim Eintreffen des Ergebnisses des Aufrufs informiert. Der Network Scheduler entnimmt dem Operation Log je nach Verbindungsstatus Aufrufe und sendet diese an den Zielrechner. In Rover können Prioritäten für Aufrufe innerhalb einer Anwendung sowie zwischen mehreren Anwendungen vergeben werden. Anhand der Prioritäten kann der Network Scheduler die Reihenfolge der Aufrufe ändern. Außerdem können mehrere Aufrufe mit dem gleichen Ziel gruppiert und in einer Nachricht übertragen werden. Beim Server werden Zustände der Aufrufverarbeitung nach Erhalt des Aufrufes, beim Beginn der Ausführung und nach Beenden der Ausführung gespeichert [JoK96], um ein schnelles Wiederanlaufen der Verarbeitung nach Fehlern und Systemabstürzen zu ermöglichen. Auf Clientseite werden QRPC-Anfragen erst aus dem Log entfernt, wenn eine entsprechende Antwort eintrifft. QRPCs unterstützen eine at-most-once Fehlersemantik.

Adaption von Protokollen auf der Middleware- und Anwendungsschicht

In [SKS+99] und [SSZ01] wird eine Plattform zur Mobilitätsunterstützung beschrieben. Kern der Plattform ist ein verteilter Queuing Dienst, der eine zuverlässige Übertragung von Anwendungsdaten ermöglicht. Dieser vermittelt Anwendungsdaten nach dem Store-and-Forward Prinzip zu bzw. zwischen mobilen Rechnern. Die Daten sind damit vor Verlust durch Verbindungsabbrüche geschützt. Außerdem agiert der Dienst während Abkopplungsphasen als Stellvertreter für mobile Endgeräte und ermöglicht die Auslieferung von Daten nach einem erneuten Verbindungsaufbau. Die Plattform unterstützt weiterhin die Adaption der Anwendungsdaten und stellt dazu Informationen über den Kommunikationspfad zum mobilen Endgerät zur Verfügung.

In [BIP+99] wird die Erweiterung von FTP um Mechanismen zur Unterstützung heterogener Netzwerkverbindungen und von Verbindungsunterbrechungen beschrieben. Der FTP-Server wurde um die Möglichkeit einer inkrementellen Datenübertragung erweitert. Eine Übertragung kann damit nach einer Unterbrechung auf den Stand vor der Unterbrechung aufsetzen. Dateianforderungen können von Anwendungen mit einer Beschreibung der Mindestanforderungen an das Netzwerk (z. B. ein Minimum für die Datenrate und ein Maximum für die Übertragungskosten) sowie einer Priorität versehen werden. Entsprechend dieser Angaben werden Anforderungen in einer Prioritätswarteschlange eingereiht und von einem Scheduler ausgeführt. Dieser startet eine Datenübertragung, wenn die verfügbare Netzwerkverbindung den Mindestanforderungen der Anfrage mit der höchsten Priorität entspricht. Wird während der Übertragung eine bessere Verbindung verfügbar, wird die Übertragung unterbrochen und über die bessere Verbindung fortgesetzt. Sinken die Eigenschaften der Netzwerkverbindung unter die Mindestanforderungen, wird die Übertragung angehalten, bis die Anforderungen wieder von einer verfügbaren Verbindung erfüllt werden können. Damit können während der Übertragung frei werdende Ressourcen für laufende Übertragungen mit benutzt werden. Außerdem können Verbindungsunterbrechungen behandelt und teuere bzw. langsame Übertragungen vermieden werden.

2.1.3 Mechanismen zur Adaption von Anwendungsdaten

Die bisher diskutierten Mechanismen zielten vorrangig auf die Eigenschaften drahtloser Kommunikationsverbindungen. Wesentliche Ansätze sind die Erhöhung der Autonomie leistungsfähiger tragbarer Rechner, die Verlagerung von Verarbeitungsschritten auf Festnetzrechner, die Behandlung von Verbindungsunterbrechungen und Abkopplungen sowie angepasste Kommunikationsprotokolle insbesondere auf Schicht 3, 4 und 5 des ISO/OSI-Referenzmodells [Tan97]. Bei diesen Ansätzen werden vorhandene Ressourcen auf dem Endgerät bzw. auf Rechnern im Internet genutzt, um die Interaktionen über schmalbandige und fehleranfällige Kommunikationskanäle zu reduzieren bzw. bei Verbindungsunterbrechungen zu verzögern. Dazu wurde vor allem Speicher- (z. B. für Caching und Queueing) und Rechenleistung (z. B. zur lokalen Verarbeitung auf dem Endgerät bzw. im Festnetz) eingesetzt, um die Datenmenge bzw. den Zeitpunkt von Interaktionen anzupassen. Damit kann der Datenaustausch in heterogenen Netzwerken erheblich verbessert werden. Insbesondere angepasste Kommunikationsprotokolle und eine dynamisch angepasste Balance zwischen lokaler Verarbeitung und Netzwerkkommunikation führen zu wesentlichen Leistungssteigerungen mobiler Anwendungen.

Ein weiterer bedeutsamer Punkt der Datenübertragung ist die Datenmenge. Die bisher beschriebenen Ansätze zielten auf die Fragestellung, wann Daten übertragen werden sollen, wie diese übertragen werden sollen und wo diese zwischengespeichert werden können. Nachfolgend sollen deshalb Ansätze vorgestellt werden, die eine Adaption von Anwendungsdaten, entsprechend der Fragestellung, welche Daten übertragen werden sollen, zum Ziel haben. Ebenso von Bedeutung wie die Anpassung der Datenmenge an den Kommunikationskanal ist die Berücksichtigung von Endgeräteeigenschaften. Auch diese können eine Anpassung der Anwendungsdaten erfordern, um die Fähigkeiten des jeweiligen Endgerätes optimal zu unterstützen bzw. Anwendungen auf Geräten mit geringen Ressourcen überhaupt erst sinnvoll zu ermöglichen.

2.1.3.1 Voradaptierte Daten

Die einfachste Variante der Datenadaption ist die Bereitstellung von Datenobjekten in mehreren Qualitätsstufen. Die Adaption wird zur Entwicklungszeit ausgeführt. Damit wird

zur Laufzeit keine Rechenleistung für die Adaption mehr benötigt. Entsprechend muss aber wesentlich mehr Speicherplatz für die Anwendungsdaten zur Verfügung gestellt werden. Dies kann insbesondere bei umfangreichen Daten wie Audio oder Video zu Engpässen führen bzw. die verfügbaren Qualitätsstufen einschränken. Auf Basis der voradaptierten Daten kann die Anwendung entsprechend der Ausführungsumgebung zur Laufzeit eine Version auswählen bzw. bei Änderungen der Ausführungsumgebung auf eine andere Version zugreifen. Die Flexibilität der Adaption ist damit auf die zur Entwicklungszeit erstellten Varianten eingeschränkt.

Odyssey [NSN97] benutzt diesen Ansatz, um adaptive Anwendungen zu realisieren. Das System verwaltet die Ressourcen mobiler Endgeräte und ermöglicht es Anwendungen durch die Erweiterung von Systemaufrufen, Bedingungen über den Zugriff auf Ressourcen zu definieren. Odyssey hat die Aufgabe, die Ressourcen des Endgerätes zu überwachen und Anwendungen über Änderungen zu informieren. Das Ressourcenmanagement ist vorrangig auf die Netzwerkbandbreite ausgerichtet. Anwendungen greifen über ein virtuelles Dateisystem auf entfernte Daten zu und registrieren für die Verbindung Anforderungen an die verfügbare Bandbreite durch eine obere und untere Grenze. Bei Änderungen der Ressourcen über diese Grenzen benachrichtigt Odyssey die Anwendung. Diese adaptiert dann ihr Verhalten beispielsweise durch den Zugriff auf alternative Daten geringerer Qualität. So liegen in einer Beispielrealisierung verschiedene Videodaten in mehreren Qualitätsstufen auf dem Server bereit. Die Anwendung wählt dann entsprechend der Ressourcenverfügbarkeit die Version mit der passenden Qualität aus. Bei Änderungen kann die Übertragung des Videos in einer anderen Qualitätsstufe fortgesetzt werden.

2.1.3.2 Filterung von Daten

Eine Erzeugung der adaptierten Daten zur Laufzeit ist im Gegensatz zu einer Erstellung zur Entwicklungszeit wesentlich flexibler. Auf Änderungen der Ausführungsumgebung kann dynamisch reagiert werden. Insbesondere wird auch die Unterstützung von Anforderungen ermöglicht, die zur Entwicklungszeit nicht betrachtet oder nicht vorhersehbar waren (z. B. neue Geräteklassen und Netzwerktechnologien). Die Filterung stellt dafür einen mächtigen Ansatz zur Verfügung, der einerseits eine wesentliche Reduzierung der Datenmenge erreichen kann, andererseits aber auch eine fein-granulare Steuerung der Datenreduzierung zulässt. Außerdem benötigen Filter in der Regel wenig Rechenzeit, da sie Daten nur nach definierten Regeln auswählen und verwerfen müssen.

Filtermechanismen können auf allen Ebenen der Systemarchitektur eingesetzt werden. In [ZeD95, Zen99] wird ein Ansatz zur anwendungsabhängigen Filterung der Kommunikation vom und zum mobilen Endgerät auf einem Stellvertreter im Festnetz beschrieben. Der Stellvertreter stellt eine generische Ausführungsumgebung zur Verfügung, in die anwendungsspezifische und protokollabhängige Filter fest oder dynamisch zur Laufzeit installiert werden können. Es werden zwei Arten von Filtern unterschieden:

1. High Level Filter für Protokolle oberhalb der Socketschnittstelle (z. B. HTTP, MPEG-Video, SMTP) und
2. Low-Level Filter für Protokolle, die unterhalb der Socketschnittstelle arbeiten (ICMP, TCP, UDP).

Durch die Installation von Filtern im Stellvertreter können optimierte Transportprotokolle (z. B. das Snoop-Protokoll als TCP-Erweiterung für die mobile Kommunikation [BSA+95]) zwischen mobilem Endgerät und Stellvertreter verwendet werden. Dies kann transparent für Kommunikationspartner im Festnetz erfolgen. Außerdem können Datenpa-

kete vor der Übertragung zum Endgerät verlustfrei komprimiert (z. B. Text in HTTP-Dokumenten oder Dateien), durch Filterung verworfen (z. B. Frames eines MPEG-Stromes) oder verzögert werden (z. B. E-Mail-Nachrichten). Für die verlustfreie Kompression der Daten ist neben dem Filter im Festnetz ein zweiter Filter zur Dekompression auf dem Endgerät notwendig. Durch diese Mechanismen kann die über die letzte Verbindung zum Endgerät übertragene Datenmenge reduziert und damit eine bessere Übertragungsleistung sowie geringere Kosten erreicht werden. Der Stellvertreter bietet weiterhin die Möglichkeit, im Festnetz für das mobile Endgerät zu agieren, indem er z. B. ICMP-ECHO Nachrichten stellvertretend für das Endgerät beantwortet, solange dieses für den Stellvertreter erreichbar ist.

In [SpS00] wird ein Ansatz zur Filterung von E-Mail Nachrichten anhand verschiedener Kriterien beschrieben. Vor der Übertragung werden Nachrichten mit Hilfe einer benutzerkonfigurierten Liste von Kriterien anhand verschiedener Informationen im Nachrichtenkopf ausgewählt. Es werden nur Nachrichten weitergeleitet, die mindestens eines der definierten Kriterien erfüllen. Nach dieser grob-granularen Auswahl vollständiger Nachrichten, werden die Nachrichten in die Bestandteile: Nachrichtenkopf, Text sowie die einzelnen Anhänge zerlegt. Diese Bestandteile können dann fein-granular gefiltert werden. So können einzelne Einträge des Nachrichtenkopfes zu einem verkürzten Nachrichtenkopf zusammengefasst werden. Die Attachments werden anhand ihres Datentyps sowie eines vorgegebenen Schwellwertes für diesen Typ gefiltert. Der Text wird ohne Veränderung übertragen. Damit ist eine sehr fein-granulare Filterung von Nachrichten und deren Inhalt möglich.

2.1.3.3 Adaption von Mediendaten

Die Filterung unterstützt nur eine Auswahl bestimmter Informationen, die dann vollständig verworfen werden. Meist wird aber eine Adaption angestrebt, die einen möglichst großen Teil der in den Originaldaten enthaltenen Informationen beibehält. Eine Vielzahl von Mechanismen wurde im Bereich Multimedia untersucht, um die Datenmenge von Medien (vor allem Audio und Video) bei möglichst geringem Informationsverlust zu reduzieren. Dazu werden sogenannte verlustbehaftete Kompressionsverfahren eingesetzt, die hybride Kompressionsverfahren darstellen. Diese basieren auf einer Kombination von verlustfreien (Entropiekodierung) und in der Regel verlustbehafteten Kodierungsverfahren (Quellenkodierung) [Ste99]. Die Entropiekodierung arbeitet verlustfrei und kann auf beliebige Daten angewendet werden. Die Quellenkodierung nutzt dagegen die Semantik der zu kodierenden Informationen und medienspezifische Eigenschaften bezüglich der Wahrnehmung durch Menschen, um die Datenmenge mit möglichst geringen Informationsverlusten zu reduzieren.

Kompressionsverfahren

Das allgemeine Vorgehen bei der Kompression von Einzelbildern sowie Audio- und Videoströmen besteht aus vier aufeinander folgenden Schritten [Ste99]. Im Schritt der *Datenaufbereitung* wird eine geeignete digitale Repräsentation des zu verarbeitenden Mediums erzeugt (z. B. Blöcke zu 8x8 Pixel mit fester Anzahl von Bits pro Pixel für ein Bild). Der zweite Schritt zur *Datenverarbeitung* erfolgt in Vorbereitung auf eine Kompression, verwirft selbst jedoch noch keine Daten (z. B. die Transformation von Bilddaten vom Zeit- in den Frequenzbereich durch eine diskrete Kosinustransformation). Im dritten Schritt der *Quantisierung* wird die Genauigkeit der zuvor ermittelten Werte reduziert. Im vierten Schritt werden dann auf diese Daten Verfahren zur *Entropiekodierung* angewendet, um

diese zusätzlich verlustfrei zu komprimieren. Das beschriebene Vorgehen wird in Abbildung 2-6 dargestellt.

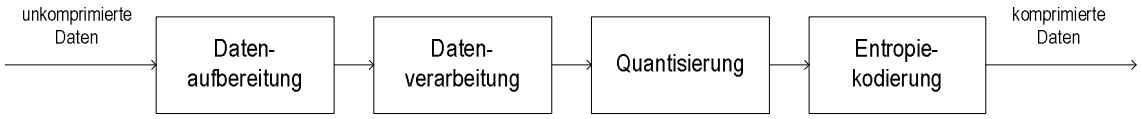


Abbildung 2-6: Wesentliche Schritte zur Kompression von Mediendaten (nach [Ste99])

Die Schritte zur Bildverarbeitung und Quantisierung können auf Basis der gleichen Verfahren oder durch die Kombination unterschiedlicher Verfahren mehrfach durchlaufen werden. Wichtige Verfahren zur Entropiekodierung sind die Lauflängenkodierung [Gol66], die Huffman-Kodierung [Huf52] und die arithmetische Kodierung [Lan84]. Quantisierungsverfahren sind die diskrete Kosinus- und die schnelle Fourier-Transformation, die relative Kodierung (z. B. Differential Pulse Code Modulation (DPCM)) und adaptive Kompressionsverfahren, wie adaptive DPCM. Wichtige Standards zur Bild-, Audio- und Videokodierung bauen auf diesen Verfahren auf. Dies sind z. B. JPEG (Joint Photographics Expert Group) und MPEG (Motion Pictures Expert Group). Für weitere Informationen zu dieser Thematik sei unter anderem auf [Ste99] verwiesen.

Konvertierungsverfahren

Kompressionsverfahren arbeiten auf der Ebene der Kodierung von Mediendaten. Sie verändern deren Informationsgehalt, behalten aber die weiteren Eigenschaften der Medien, wie Auflösung, Samplerate oder Farbtiefe bei. Konvertierungsverfahren ändern dagegen gezielt diese Merkmale. Beispielsweise wird im Pythia-Projekt [FoB96, FGC+98] ein Stellvertreter im Festnetz eingesetzt, um die zu übertragende Datenmenge zum mobilen Endgerät zu reduzieren. Pythia ist ein als HTTP-Proxy [LuA94] implementiertes System zur Adaption von Anwendungsdaten zur Laufzeit. Der Ansatz verfolgt das Ziel der Unterstützung der Heterogenität der Endgeräte bezüglich Hard- und Software sowie der variierenden Dienstgüte von Netzwerkverbindungen. Insbesondere soll die Ende-zu-Ende-Verzögerung beim Zugriff auf Daten reduziert werden. Dazu werden datentypabhängige Verfahren zur Reduzierung der zu übertragenden Datenmenge eingesetzt. Dies sind zum einen verlustbehaftete Kompressionsverfahren für Bilder, zum anderen Mechanismen zur nachträglichen Anforderung und Darstellung von Teildaten mit höherer Qualität (z. B. ein Bildausschnitt in der Qualität des Originals). Außerdem wird die Konvertierung von Postscript-Dokumenten in HTML und RTF unterstützt. Durch datentypabhängige Verfahren zur verlustbehafteten Kompression können die Spezifika einzelner Datentypen ausgenutzt werden, um eine Reduzierung der Datenmenge bei minimalen Verlusten von Informationen zu erreichen. Die Anpassung erfolgt an eine vom Benutzer festgelegte maximale Verzögerungszeit sowie an die verfügbare Datenrate und Displaygröße des Endgeräts. Diese Informationen werden dem Proxy über ein Profil zur Verfügung gestellt. Der Ansatz ermöglicht insbesondere die Darstellung von Daten auf ressourcenarmen Endgeräten, die im Original auf diesem nicht darstellbar wären.

2.1.3.4 Adaption web-basierter Benutzerschnittstellen

Adaptive Web-Anwendungen sind Gegenstand einer Reihe von Untersuchungen, die teilweise sehr verschiedene Ansätze zur Realisierung adaptiver Dienste aufzeigen. Viele der Arbeiten verwenden HTML als Quellsprache. Dies ermöglicht einen adaptiven Zugriff auf alle bereits existierenden Web-Dokumente in HTML, ohne dass diese geändert werden

müssen. In der Praxis werden in vielen HTML-Dokumenten aber Inhalt und Struktur miteinander vermischt. Informationen über die Dokumentstruktur werden in diesen Ansätzen mit Hilfe von Heuristiken gewonnen. Die aus der Adaption resultierenden Seiten sind deshalb in Bezug auf Design und Benutzbarkeit oft unbefriedigend und teilweise für bestimmte Geräteklassen unbenutzbar.

Digestor [BGS99] ist ein Beispiel für diesen Ansatz. Anhand von Tag-Informationen und Heuristiken wird die Struktur des Dokumentes extrahiert. Diese Informationen werden zur Restrukturierung von Text und Tabellen, zur Erstellung von Übersichtsseiten, zur Adaption von Bildern und zur Filterung von Inhalten verwendet. M-Links [STH+01] zielt auf die Änderung der Dokumentstruktur, um insbesondere Geräte mit kleinen Displays zu unterstützen. Der Webzugriff wird in die zwei Phasen Navigation und Dienstnutzung unterteilt. Die Dokumentstruktur wird in eine Liste von Verweisen geändert, die zum Teil durch Heuristiken aus dem Dokument extrahiert werden. Beispielsweise wird eine Telefonnummer in einen Verweis transformiert, der mit einem Dienst zum Wählen der Nummer verknüpft ist. In [BGP01] wird strukturierter Text in Segmente zerlegt, die mit unterschiedlich umfangreichem Inhalt angezeigt werden können. M-Links und [BGP01] basieren ebenfalls auf HTML als Quellsprache und zielen auf die Verbesserung der Navigation für Geräte mit kleinen Displays. [BGP01] zielt vorwiegend auf Geräte mit Stifteingabe.

Ansätze deren Quellsprachen die Dokumentstruktur explizit beschreiben erlauben eine bessere Steuerung der Adaption und resultieren deshalb in Seiten mit besserem Design und besserer Benutzbarkeit. Durch das Hinzufügen von Meta-Informationen kann außerdem die Adaption gezielt gesteuert werden, was die Resultate weiter verbessert. In Wit [Wat94b] werden Webdokumente als verknüpfte Hyperobjekte beschrieben. Die Struktur der Hyperobjekte reflektiert die hierarchische Struktur der Dokumente (z. B. wird strukturierter Text als eine Menge miteinander verknüpfter Hyperobjekte dargestellt). Den Verknüpfungen können Meta-Informationen wie eine relative Priorität oder die Zugriffswahrscheinlichkeit hinzugefügt werden. Mit diesen Informationen werden Caching, Prefetching und die Datenreduzierung gesteuert. Wit adressiert damit vor allem die Anpassung der Kommunikation über drahtlose Netzwerke und unterstützt ausschließlich Palm PDAs.

Top Gun Wingman [FGG+98] basiert ebenfalls auf dem Proxy-Konzept. Auf einem Proxy im Festnetz können Adaptionsmodule mit klar definierten Aufgaben installiert werden. Implementiert wurden Module zur Transformation von Bildern, HTML-Dokumenten und ZIP-Archiven. Der Browser unterstützt jedoch nur eine spezielle Klasse von PDAs und enthält damit keine Konzepte zur Unterstützung beliebiger Endgeräte.

iRoom [JFW02] ist eine Meeting-Anwendung, die konzentriert auf einen Raum Informationen über Geräte, Personen und Anwendungen in diesem Raum nutzt, um eine Integration von Anwendungen und eine Kooperation zwischen den Personen im Raum zu erreichen. Insbesondere können große Displays, die im Raum installiert wurden, genutzt werden, um Daten und Anwendungen von beliebigen Geräten darzustellen. Die Steuerung von Geräten und Anwendungen kann dabei von jedem Gerät im Raum ausgehen und dynamisch übergeben werden. Basis von iRoom ist das Interactive Room Operating System (iROS), das als Integrationsschicht (Meta-OS) auf allen Geräten installiert wird. Teil der iROS Plattform ist die Komponente ICrafter [PLF+01], die Benutzerschnittstellen für Dienste in iRoom erzeugen, anpassen und integrieren kann. Dazu können Generatoren installiert werden, die ausgehend von einer Dienstbeschreibung für eine bestimmte Geräteklasse Benutzerschnittstellen erzeugen und anpassen können. Generatoren können dienstspezifisch oder generisch für eine bestimmte Geräteklasse arbeiten. Außerdem wird die Aggre-

gation von Diensten unterstützt, indem mehrere Dienstbeschreibungen durch entsprechende Generatoren in eine gemeinsame Benutzerschnittstelle integriert werden.

2.1.3.5 Meta-Informationen zur Steuerung der Adaption

Eine vollständig automatische Adaption zur Laufzeit kann nur die ihr zur Verfügung stehenden Informationen über die Anwendung und weiteren Kontext einbeziehen. Für viele Adaptionsmechanismen ermöglicht aber eine Unterstützung durch den Anwendungsentwickler bzw. Benutzer bessere Ergebnisse im Vergleich zu einer rein kontextbasierten Adaption. Dies gilt insbesondere für die Adaption von Benutzerschnittstellen. Eine manuelle Adaption führt für dieses Anwendungsgebiet der Adaption hinsichtlich Design und Benutzbarkeit zu wesentlich besseren Ergebnissen. Im Vergleich zu einer automatischen Adaption ist eine manuelle Adaption jedoch zu aufwendig und selten ökonomisch zu rechtfertigen. Die Kombination manueller und automatischer Adaption ermöglicht durch das Bereitstellen manuell erzeugter Meta-Informationen zur Entwicklungszeit die Steuerung der automatischen Adaption zur Laufzeit [SpG02]. Dies führt zu einem vertretbaren Kompromiss zwischen Entwicklungsaufwand sowie Design und Benutzbarkeit.

In [MVK+02] wird ein Verfahren vorgestellt, welches die Erstellung adaptiver Webanwendungen basierend auf XHTML ermöglicht. In das Quelldokument werden zusätzliche Tags zur Steuerung der Adaption eingefügt (Adaptive-User-Interface-Tag-Library, AUI). Mit Hilfe der AUI können selektive Einfügungen und Auslassungen, die Zerlegung der Darstellung eines Dokumentes bzw. einer Anwendung in Teildokumente/-anwendungen (Pagination), strukturelle Transformation und „Pass-through“ Funktionalität zur direkten Einbindung von Zielsprachen-Markupcode umgesetzt werden.

Ein Verfahren zur Umsetzung dokumentextern spezifizierter Annotationen zur semantischen Dialogadaption wird in [HKO+00] beschrieben. Die zur automatisierten Adaption notwendigen Informationen über die Dialogsemantik werden in einer vom eigentlichen Dialog getrennten Beschreibung gehalten. Dabei können Alternativrepräsentationen, Hinweise zur Pagination und prioritätsbasierte Auswahlkriterien zur selektiven Auslassung spezifiziert werden.

In [SpG02] wird eine generische Beschreibungssprache für Web-Anwendungen mit dem Konzept der Meta-Informationen beschrieben. Meta-Informationen können zur Entwicklungszeit gemeinsam mit den Inhalten und der Struktur vom Entwickler erstellt werden. Die Informationen werden dann zur Laufzeit ausgewertet und in die automatische Adaption einbezogen. So können alternative Dokumentbereiche, ein angepasstes Layout, angepasste Inhalte sowie eine Unterstützung der Fragmentierung von Web-Dokumenten kontextabhängig definiert werden.

2.1.4 Zusammenfassung

Die beschriebenen Projekte und Ansätze untersuchen eine Vielzahl von Mechanismen zur Unterstützung mobiler Infrastrukturen. Die einzelnen Lösungen fokussieren dabei auf bestimmte Problemstellungen. Ein wesentliches Problem entsteht durch drahtlose Verbindungen, die durch eine geringe Datenrate und hohe Verzögerungszeiten bei gleichzeitig hohen Fehlerraten und hohen Kommunikationskosten charakterisiert sind. Abgekoppelte und autonome Operationen zielen deshalb auf eine ausgewogene Balance zwischen lokaler Verarbeitung und Netzwerkkommunikation. Abgekoppelte Operationen erhöhen die Autonomie gegenüber der Netzwerkverbindung, setzen dafür aber Ressourcen des Endgeräts ein, indem die Verarbeitung lokal auf dem Endgerät stattfindet. Wesentliche Mechanismen

sind Caching, Vorabladen und eine lokale Verarbeitung zur Simulation der Serverfunktionalität. Autonome Operationen führen dagegen Code auf Rechnern im Festnetz aus und übermitteln ihre Ergebnisse, wenn eine Verbindung zum mobilen Endgerät besteht. Da aber in der Regel anwendungsspezifischer Code ausgeführt wird, werden Mechanismen zur dynamischen Codeinstallation bzw. für mobilen Code benötigt. Erreicht wird bei beiden Ansätzen eine Unabhängigkeit von der Netzverbindung. Entsprechend der verfügbaren Verarbeitungsressourcen auf dem Endgerät sowie der Leistungsfähigkeit der Netzwerkverbindung kann mit diesen Mechanismen eine Balance zwischen der Nutzung von Ressourcen zur Verarbeitung sowie zur Kommunikation hergestellt werden. Eine Verzögerung der Datenübertragung durch Lazy Evaluation und Delayed Write-Back ermöglicht dabei eine über die Zeit verteilte, gleichmäßige Nutzung von Kommunikationsressourcen. Durch eine lokale Verarbeitung kann zusätzlich eine Reduzierung von Interaktionen erreicht werden (z. B. durch Zusammenfassen von Nachrichten oder Operationen). Damit wird das Problem leistungsschwacher Kommunikationskanäle umgangen. Beantwortet wurden vor allem die Fragen „wann“ Daten übertragen werden sollen und „wo“ diese zwischengespeichert werden können, um Übertragungen zu vermeiden.

Angepasste Kommunikationsprotokolle zielen dagegen direkt auf eine bessere Ausnutzung der Ressourcen drahtloser Kommunikationskanäle, die durch Annahmen in Standardprotokollen für drahtgebundene Netze nicht genutzt werden können. Damit werden drahtlose Verbindungen weit weniger zu einem Engpass für Anwendungen. Mechanismen zur Fehlerkorrektur, zur Auftrennung von Verbindungen und zur differenzierten Behandlung von Paketverlusten sowie angepasste Protokolle für bestimmte Übertragungsmedien zielen vor allem auf die Frage, „wie“ Daten übertragen werden.

Die Anpassung der Anwendungsdaten zielt auf die Frage, welche Daten übertragen werden sollen. Damit wird die Datenmenge sowie die Qualität von Anwendungen beeinflusst. Durch eine Filterung, aber auch durch ein Ersetzen oder Transformieren können Anwendungsdaten an die verfügbare Datenrate angepasst werden. Der Verlust an Information oder Dienstqualität ist dabei abhängig vom ausgewählten Mechanismus. Beispielsweise resultieren verlustbehaftete Kompressionsverfahren in einem wesentlich geringeren Qualitätsverlust bei gleicher Reduzierung der Datengröße als eine Filterung oder Konvertierung der Daten (z. B. Skalierung eines Bildes). Die Auswahl entsprechender Mechanismen zur Adaption hat somit einen großen Einfluss auf die Qualität einer der Anwendungsdaten.

2.2 Kombination von Adaptionsmechanismen

Die bisher vorgestellten Projekte hatten die Untersuchung einzelner Adaptionsmechanismen zum Ziel. Einige Lösungen kombinieren zwar auch Mechanismen miteinander, realisieren dies jedoch in spezifischer Form ohne eine Betrachtung generischer Aspekte sowie der Wiederverwendung. Dies erfolgt im Rahmen von Lösungen für bestimmte Anwendungsfelder oder begrenzt auf bestimmte Mechanismen.

Die im Folgenden beschriebenen Lösungen betrachten dagegen die Kombination von Mechanismen in generischer Weise. Mechanismen werden als autonome Filter bzw. Komponenten betrachtet, die zu Pfaden bzw. Graphen kombiniert werden können. Die betrachteten Ansätze wurden in Lösungen zur Adaption von Medienobjekten und Kommunikationsprotokollen sowie zur dynamischen Erzeugung verteilter Komponentenpfade unterteilt.

2.2.1 Filterpfade bzw. –graphen zur Adaption von Medienobjekten

Die im weiteren Verlauf betrachteten Ansätze stammen aus dem Datenbankbereich und untersuchen die Kombination von Filtern zur Medientransformation. Zur Kombination der Filter wird deren Eigenschaft ausgenutzt, dass Filterfunktionen durch die Datentypen der ein- bzw- ausgegebenen Medienobjekte charakterisiert werden. Unter dieser Annahme können Filter automatisiert zu Pfaden bzw. Graphen zusammengesetzt werden, die ausgehend von einem existierenden Medienobjekt das angeforderte Medienobjekt erzeugen.

2.2.1.1 Synthesesequenzen für Medienobjekte

Ein Ansatz zur Erzeugung von Synthesesequenzen für Medienobjekte in einer verteilten, kooperativen Umgebung wird in [CRS98] beschrieben. Ausgangspunkt sind die formalen Definitionen eines kollaborativen Multimediasystems sowie eines Medienobjekts. Ein Medienobjekt o wird durch seinen Datentyp *ds(o)* (z. B. GIF oder Postscript), seinen Namen *name(o)* in der Form *<string>.<type>*, seine Größe *size(o)*, seinen Objekttyp *ot(o)* und seine Objektcharakteristik *oc(o)* beschrieben. Der Objekttyp charakterisiert die Zeitabhängigkeit der Repräsentation des Medienobjektes und ist ein Element der Menge {*temporal*, *quasi-temporal*, *static*, *quasi-static*}. Die Objektcharakteristik ist abhängig vom Objekttyp und beschreibt die Anzahl der Informationsblöcke und deren Anzeigedauer.

Ein kollaboratives Multimediasystem COMS $\Gamma=(G, Obj, loc, HC, CAP)$ besteht aus einer Menge vernetzter Knoten, repräsentiert durch einen ungerichteten, gewichteten Graphen $G=(V,E, \rho)$ mit der Knotenmenge *V*, der Menge *E* von Kanten zwischen den Knoten und der Kostenfunktion $\rho: E \rightarrow R^+$. Diese spezifiziert die Kosten für das Senden eines Bytes über eine Kante des Graphen. Verteilt auf den Knoten des Systems befinden sich eine Menge von Medienobjekten *Obj* sowie eine Menge von Medienfiltern (capabilities) zur Verarbeitung von Medienobjekten *HC*. Die Funktion $loc: Obj \rightarrow V$ spezifiziert die Menge von Knoten, auf denen sich ein Medienobjekt befindet. Die Funktion $CAP: V \rightarrow 2^{HC}$ ordnet die Menge der vorhandenen Medienfilter den Knoten des Systems zu. Medienfilter werden durch einen Namen sowie ihren Ein- und Ausgabetyp definiert und können in Medienfilter zur Anzeige (*display capability*), Manipulation (*edit capability*) sowie zur Konvertierung des Formates von Medienobjekten (*conversion capability*) unterschieden werden. Ein Medienfilter c kann nur auf ein Medienobjekt o angewendet werden, wenn dessen Eingabetyp mit dem Typ von o übereinstimmt. Ist c ein Konvertierungsfilter, dann ist das Resultat der Anwendung des Filters c auf o gleich o' ($c(o)=o'$).

Die Aufgabe des kollaborativen Multimediasystems ist es, für einen Benutzer auf Knoten A, der eine Datei von Knoten B bearbeiten will, diese direkt oder indirekt über andere Knoten in ein Format zu konvertieren, das A verarbeiten kann. Nach der Verarbeitung muss die Datei zum Knoten B zurückgesendet werden. Dabei ist eine direkte oder indirekte Konvertierung in das ursprüngliche Format notwendig. Außerdem soll das Medienobjekt die von A geforderte Qualität besitzen, wobei die Kosten für die verteilte Konvertierung und Kommunikation zu minimieren sind. Zur Lösung des Problems werden in [CRS98] Algorithmen zur Erzeugung von Synthesesequenzen entwickelt. Synthesesequenzen realisieren die für die Kooperation notwendige Übertragung und Konvertierung von Medienobjekten unter Berücksichtigung der verteilt verfügbaren Medienobjekte und Medienfilter. Außerdem bieten die Algorithmen einen Rahmen zur Assoziierung von Kosten mit Synthesesequenzen und die Suche einer optimalen Sequenz in Bezug auf diese definierten Kosten.

Ein Medienobjekt o_m kann demnach auf dem Knoten N_m erzeugt werden, wenn es ein Medienobjekt o_1 auf einem Knoten N_1 und eine Folge von Konvertierungen und Übertragungen gibt, die das Medienobjekt o_m auf dem Knoten N_m erzeugen. Notiert wird die verteilte Synthesesequenz für das Objekt o_m für den Knoten N_m mit:

$$(N_1, o_1) \xrightarrow{c_1} (N_2, o_2) \xrightarrow{c_2} (N_3, o_3) \xrightarrow{c_3} \ldots (N_{m-1} o_{m-1}) \xrightarrow{c_{m-1}} (N_m, o_m)$$

Der Algorithmus DOptOSA erzeugt für ein gegebenes kollaboratives Multimediasystem, einen Zielknoten N und ein Medienobjekt o mit vorgegebenem Format eine optimale verteilte Synthesesequenz. Der Ausgangspunkt ist die Menge von Medienobjekten im System, die sich durch ihr Format, nicht jedoch durch ihren Inhalt vom Zielobjekt unterscheiden. Ausgehend von dieser Menge wird eine geordnete Liste von initialen Synthesesequenzen erzeugt. Für die weiteren Schritte wird jeweils die erste Sequenz in der Liste, d. h. die am besten bewertete Sequenz, ausgewählt. Von dieser werden alle Synthesesequenzen abgeleitet, die sich durch Anwendung eines lokalen Konvertierungsfilters oder der Sendeoperation zu einem neuen Knoten erzeugen lassen. Diese Synthesesequenzen werden auf Zyklen überprüft und unter Anwendung des Bewertungskriteriums in die Liste von Synthesesequenzen eingeordnet. Wird eine Synthesesequenz für das gesuchte Objekt gefunden, sichert der Algorithmus, dass diese Sequenz optimal hinsichtlich des Bewertungskriteriums ist, da alle folgenden Sequenzen in der Liste bereits eine schlechtere Bewertung besitzen.

Durch die Wahl von Bewertungskriterien können die Anforderungen an eine optimale Synthesesequenz festgelegt werden. Den Medienfiltern werden dafür Angaben über die Änderung der Größe eines Medienobjekts (*size ratio*) und den damit verbundenen Verarbeitungskosten (*cost ratio* pro Byte) hinzugefügt. Unter dieser Voraussetzung können die Größe des Zielobjektes sowie die gesamten Verarbeitungskosten für eine Synthesesequenz ermittelt werden. Eine weitere Funktion beschreibt die Änderung der Qualität eines Medienobjektes durch einen Medienfilter und ermöglicht die Berechnung der Qualität des Zielobjektes und damit eine Bewertung von Sequenzen hinsichtlich dieses Kriteriums. Die Einzelkriterien können mit unterschiedlicher Wichtung kombiniert werden. Dieser Ansatz stellt einen Rahmen zur Bewertung von Synthesesequenzen dar und kann durch zusätzliche Kriterien erweitert werden.

2.2.1.2 Virtual Media

VirtualMedia [Mar00, Mar01b] beschreibt ein Konzept aus dem Bereich der Datenbanken. Ziel des Konzeptes ist die Realisierung universeller Medienserver, die eine Vielzahl heterogener Clients und Benutzeranforderungen unterstützen. Die Mechanismen zur Speicherung und Verarbeitung von Medienobjekten sollen anwendungsneutral arbeiten. Dies soll durch Transformationsunabhängigkeit zwischen dem internen Speicherformat und Verarbeitungsoperationen des Datenbanksystems und den von Clients angeforderten Medienformaten und Operationen erreicht werden. Demnach kann das Datenbanksystem das interne Speicher- und Verarbeitungsformat nach Gesichtspunkten der Effizienz und Performance wählen. Die intern gespeicherten Medienobjekte werden dann entsprechend der Anforderungen von Clients verarbeitet und in ein externes Format transformiert. Dies erfolgt transparent für die Clients, das Datenbanksystem kann die internen Speicherformate frei wählen und eine Optimierung der auszuführenden Verarbeitungsoperationen nach internen Gesichtspunkten durchführen.

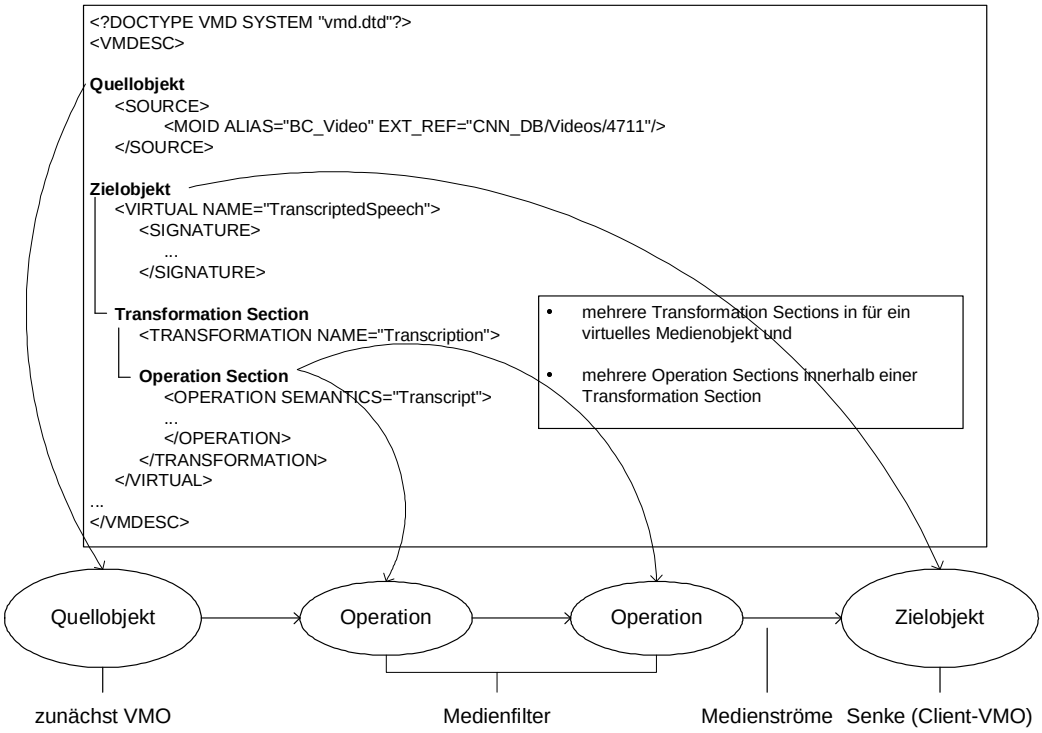


Abbildung 2-7: Erzeugung eines Transformation Request Graphen aus einer VirtualMedia Description

VirtualMedia führt zur Realisierung der Transformationsunabhängigkeit eine Abstraktionsschicht ein. Diese nimmt Anforderungen von Clients entgegen und baut einen entsprechenden Filtergraphen auf, der aus den intern gespeicherten Medienobjekten die vom Client angeforderten Medienobjekte erzeugt bzw. Transformationen dieser ausführt. Clients fordern Medienobjekte in Form von VirtualMedia Descriptions (VMD) an, die in der VirtualMedia Markup Language (VMML) formuliert werden. Eine VirtualMedia Description besteht aus einer Definition der Quell- und Zielobjekte für die Anfrage sowie Transformationsoperationen, die auf die Quellobjekte angewendet werden sollen, um die Zielobjekte zu erzeugen (siehe Abbildung 2-7). Quellobjekte sind dem Datenbanksystem bekannte Objekte und werden anhand ihres Identifikators spezifiziert. Zielobjekte werden als virtuelle Objekte anhand von Typinformationen und einer optionalen Qualitätsbeschreibung definiert. Außerdem enthält jede Beschreibung eines Zielobjektes mindestens ein „Transformation“-Element. In diesem Element wird mindestens ein Quellobjekt definiert, von dem das Zielobjekt abgeleitet werden kann. Innerhalb des „Transformation“-Elementes wird mindestens ein Operation-Element definiert, das die Operation beschreibt, mit der aus den Quellobjekten das Zielobjekt erzeugt werden soll. Eine Operation ist im einfachsten Fall eine leere Operation.

Der VirtualMedia Descriptor enthält damit zwei Anforderungen. Zum einen müssen die Quellobjekte durch das Datenbanksystem materialisiert werden, zum anderen müssen Medienfilter gefunden werden, die die angeforderten Transformationen ausführen können, um die Zielobjekte zu erzeugen. VirtualMedia verwendet zur Verarbeitung dieser Anforderungen sogenannte VirtualMedia Filter Graphen, die sowohl virtuelle Elemente (Medienobjekte und –filter) als auch reale Elemente (materialisierte Medienobjekte und instantiierbare Medienfilter) enthalten können. Für die beiden oben genannten Anforderungen werden zunächst zwei unabhängige Graphen erstellt.

Materialisierungsgraphen beschreiben die Erzeugung der Quellobjekte aus intern gespeicherten Medienobjekten. Die Graphen enthalten materialisierte Medienobjekte, die durch instantiierbare Filter in die Quellobjekte der VirtualMedia Description transformiert werden. Dabei können drei Typen von Materialisierungen unterschieden werden. (1) Primäre Materialisierungen sind die im Datenbanksystem gespeicherten Medienobjekte. (2) Sekundäre Materialisierungen werden vom Datenbanksystem zum Zweck der Verarbeitung und Optimierung erzeugt und sind transparent für Clients. (3) Abgeleitete Materialisierungen werden aus den beiden ersten Materialisierungen im Transformation Request Graphen erzeugt.

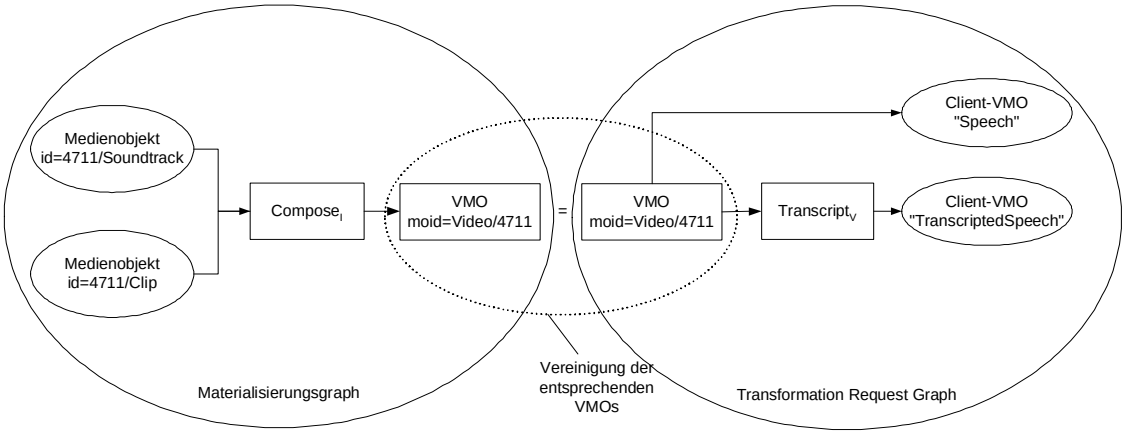


Abbildung 2-8: Vereinigung von Materialisierungsgraph und Request Transformation Graph

Transformation Request Graphen werden aus der *VirtualMedia Description* abgeleitet und enthalten die Erzeugung der Zielobjekte aus Quellobjekten. Die Elemente des Graphen sind zunächst virtuell. Die beiden Graphen enthalten damit gleiche virtuelle Medienobjekte und können an diesen Punkten zu einem Graphen vereinigt werden (siehe Abbildung 2-8).

Der so erzeugte Graph enthält weiterhin virtuelle Elemente und muss in einen instantiierbaren Graphen transformiert werden (VirtualMedia Preset Graph). VirtualMedia definiert dazu ein Datenmodell für Medienobjekte und Medienfilter und eine Menge von Regeln zur semantikerhaltenden Transformation von Filtergraphen. Das Datenmodell definiert Typsignaturen für Medienobjekte und Medienfilter. Diese sind objekt-orientiert, basieren aber nicht auf einer Typhierarchie und Vererbung. Vielmehr werden die Attribute von Medienobjekten wie Typ, Subtyp, Encoding, usw. als Properties definiert, die zusammen eine Signatur ergeben. Widersprüchliche Zusammensetzungen werden durch die Definition von entsprechenden Beschränkungen für die Zusammensetzung verhindert. Für virtuelle und instantiierbare Filter werden durch das Datenmodell funktionale und nicht-funktionale Eigenschaften definiert. Die funktionalen Eigenschaften werden durch die Ein- und Ausgabetypen der Filter beschrieben. Nicht-funktionale Eigenschaften, wie Ressourcenbedarf, Verarbeitungskomplexität oder Qualitätsreduzierung sind im Konzept vorgesehen, wurden bisher aber nicht im Detail betrachtet [Mar01a].

Die Regeln zur Graphentransformation basieren auf einer Menge von Äquivalenzrelationen bezüglich Filtern und Signaturen von Medienobjekten. Die drei grundlegenden Äquivalenzrelationen Neutralität, Umkehrbarkeit und Vertauschbarkeit beschreiben die Änderung von Filtergraphen, nicht jedoch die Ersetzung virtueller Elemente durch reale Elemente. Regeln zur Ersetzung virtueller Filter basieren auf der Assimilationsrelation. Eine weitere Relation beschreibt die Behandlung von Filterpaaren zur Dekomposition und Komposition zusammengesetzter Medienobjekte. Die aus den Äquivalenzrelationen abge-

leiteten Regeln definieren die auf Filtergraphen anwendbaren Transformationen. Die Anwendung dieser Regeln kann potentiell in beliebigen Veränderungen insbesondere Erweiterungen des Graphen resultieren. Zur Einschränkung des Suchraumes von anwendbaren Regeln werden mit den resultierenden Filtergraphen Kosten assoziiert. In [Mar01a] wird die Anwendung unterschiedlicher Algorithmen zur effizienten Erzeugung und Optimierung instantiierbarer Filtergraphen diskutiert. Bisher wird nur die Erzeugung kompletter Implementierungen unterstützt, die Suche nach optimalen Filtergraphen ist Gegenstand weiterer Betrachtungen.

2.2.2 Dynamische Erzeugung verteilter Komponentenpfade

Die nachfolgend vorgestellten Lösungen haben eine dynamische Erzeugung von Komponentenpfaden zum Ziel. Untersucht werden vor allem Laufzeitsysteme zur Instantiierung von Pfaden für eine bestimmte Kommunikationsverbindung bzw. einen bestimmten Dienstnutzer. Weitere Schwerpunkte liegen auf der Generierung einer Pfadbeschreibung aus einer vorliegenden Anforderung, die Suche und Auswahl passender Komponenten und die Transformation der Beschreibung des zunächst logischen Pfades in einen ausführbaren Pfad aus verteilten Komponenten.

2.2.2.1 Ninja Path

Ninja Path [CMI99, GWB+01] beschreibt einen Ansatz zur Kombination von einfachen Diensten zu komplexeren Diensten durch die Erzeugung von sogenannten Pfaden. Ein Pfad ist eine Sequenz von Operatoren und Konnektoren, die einen geforderten Dienst ausführt. Operatoren sind Ninja Dienste mit strombasierter Ein- und Ausgabe. Konnektoren sind Codemodule, die strombasierte Protokolle implementieren. Diese stellen eine Transportschicht ohne Kenntnis und Zugriff auf die Struktur und Semantik der übertragenen Daten zur Verfügung (z. B. für TCP und UDP). Konnektoren führen insbesondere keine Transformation zwischen Protokollen durch, dies wird auf Basis von Operatoren realisiert.

Die Pfaderzeugung von Ninja Path erfolgt in drei Schritten. Im ersten Schritt wird aus einer initialen Anforderung eines Pfades ein logischer Pfad erzeugt. Die initiale Anforderung beschreibt einen Pfad anhand des Quell- und Zieloperators sowie weiterer Operatoren, die im Pfad enthalten sein sollen. Davon ausgehend wird ein Pfad zusammengesetzt, der den geforderten Dienst erbringt. Die einzelnen Operatoren stellen dafür eine XML-basierte Beschreibung ihrer Ein- und Ausgabetypen sowie ihrer Semantik zur Verfügung. Diese Beschreibung wird in einer XML-Datenbank innerhalb der Infrastruktur von Ninja [GWB+01] registriert. Die Suche nach passenden Operatoren erfolgt anhand der Ein- und Ausgabeformate und der Semantik. Existieren mehrere Pfade, wird der kürzeste ausgewählt [CMI99], weitere Optimierungskriterien, wie etwa in [CRS98] werden nicht betrachtet. Resultat ist ein logischer Pfad von Operatoren, die durch entsprechende Konnektoren verbunden sind.

Im zweiten Schritt erfolgt die Überführung des logischen Pfades in einen physischen Pfad. Dazu werden zu jedem logischen Operator laufende Instanzen innerhalb der Ninja Infrastruktur gesucht. Laufende Dienste innerhalb einer Base[4] können über den Service Discovery Service [GWB+01] ermittelt werden. Wurden mehrere Instanzen gefunden, erfolgt in Form einer zufälligen Auswahl einer Instanz eine einfache Lastverteilung. Durch die Wahl

[4] , Eine Base ist ein Workstation-Cluster mit einer speziellen Laufzeitumgebung für Dienste in Ninja, die über ein leistungsfähiges lokales Netz gekoppelt sind.

laufender Instanzen wird auch die Platzierung von Operatoren festgelegt. Eine Reihe von Operatoren kann auch dynamisch installiert und damit frei platziert werden. In Ninja können Operatoren nur auf Active Proxies[5] dynamisch installiert werden [GWB+01]. Außerdem werden die Codemodule der Konnektoren entsprechend der Platzierung der Operatoren im System verteilt.

Im dritten Schritt werden die Bestandteile des Pfades instantiiert und gestartet. Dem Pfad wird zunächst eine eindeutige Kennung (ID) zugeordnet. Danach wird dem Pfad pro Operator ein Thread zugewiesen. Außerdem werden die Ein- und Ausgaben der Operatoren über Konnektorinstanzen verbunden. Jeder Operator erhält zusätzlich einen Verweis auf einen Operator, der im Fehlerfall benachrichtigt werden soll. Derzeit ist dies für alle Operatoren der Quelloperator. Dieser baut in der aktuellen Implementierung [CMI99] im Fehlerfall den Pfad zunächst vollständig ab und instantiiert diesen anhand der logischen Pfadbeschreibung neu. Schlägt dies fehl, muss vom Pfadsystem ein neuer logischer Pfad erzeugt werden.

Anschließend kann der Quelloperator Daten entlang des Pfades senden. Der Pfad besitzt aber keine Ende-zu-Ende Semantik, die Konnektoren realisieren nur eine gesicherte Zustellung zwischen einzelnen Operatoren. Eine entsprechende Behandlung von Übertragungsfehlern muss deshalb durch die Quell- und Zieloperatoren ausgeführt werden. Die Platzierung der Operatoren erfolgt entsprechend der Architektur von Ninja in Knoten eines Workstation-Clusters (Base) sowie auf Active Proxies. Dabei gelten außerdem Restriktionen für die Platzierung dynamisch installierbarer Operatoren. Diese werden außerhalb der Base Infrastruktur installiert und können damit die Dienste der Base zur Verfügbarkeit, Lastverteilung und Ausfallsicherheit nur indirekt nutzen. Eine Platzierung von Operatoren auf Endgeräten ist ebenfalls nicht möglich. Eine Parametrisierung von Operatoren insbesondere durch Kontextinformationen und eine Rekonfiguration bei Änderungen der Ausführungsumgebung wurden nicht beschrieben. Die Operatoren werden anhand eines selbst definierten Typsystems durch ihre Ein- und Ausgabetypen und ihre Semantik beschrieben. Eine abstrakte Beschreibung von Pfaden im Sinne einer Architekturbeschreibungssprache (siehe Abschnitt 2.5.4) existiert jedoch nicht. Insgesamt hat Ninja die Komposition komplexer Dienste aus einfachen Diensten zum Ziel. Der Fokus des Projektes liegt auf robusten und skalierbaren Diensten in Server-Clustern im Festnetz (Bases), in die Mechanismen zur Datenadaption integriert wurden.

2.2.2.2 Conductor

Conductor [YWR+99, YRE+01] stellt eine Laufzeitumgebung für Adaptionsmodule zur Verfügung. Ziel des Ansatzes ist eine anwendungstransparente Anpassung der Übertragung von Daten über heterogene und sich dynamisch ändernde Netzwerkverbindungen. Die Dienstgüte von Anwendungen soll durch die Adaption schrittweise gesenkt bzw. erhöht werden können, um die verfügbaren Übertragungsressourcen effizient zu nutzen. Erreicht werden soll dies durch einen kombinierten und koordinierten Einsatz verteilter Adaptionsmechanismen, dem eine zentrale Planung unterliegt.

Basis des Ansatzes ist ein Laufzeitsystem, das auf einer Teilmenge von Knoten im Netzwerk installiert wird (z. B. auf Endgeräten, Servern, Gateways zwischen Netzwerktechno-

[5] Active Proxies sind spezielle Rechner außerhalb der Workstation Cluster zur Unterstützung bestimmter Klassen von Endgeräten. In der Regel sind diese an den Übergangspunkten zwischen Fest- und funknetz platziert.

logien oder Proxies). Dieses enthält einen Mechanismus zur Unterbrechung von TCP-Verbindungen während des Verbindungsaufbaues, eine Infrastruktur zur Planung der Adaption, eine Laufzeitumgebung für Adaptionsmechanismen für jeden Knoten und Mechanismen zur Fehlerbehandlung (siehe Abbildung 2-9).

Die Adaptionsmechanismen werden in Conductor in Form von Adaptoren bereitgestellt. Diese kommunizieren über strombasierte Protokolle und können beliebige Anpassungen auf Anwendungsebene ausführen. Diese kann verlustbehaftet oder verlustfrei sein. Adaptionen umfassen z. B. Protokolltransformationen in Form von Adaptorenpaaren oder die Anpassung von Bild- und Videodaten. Jeder Adaptor besitzt eine Beschreibung seines Ein- und Ausgabeprotokolls, der benötigten Ressourcen und Informationen über seine Funktion (Auswirkungen seiner Anwendung z. B. auf die Komprimierbarkeit der Daten) und die Kombinierbarkeit mit anderen Adaptoren.

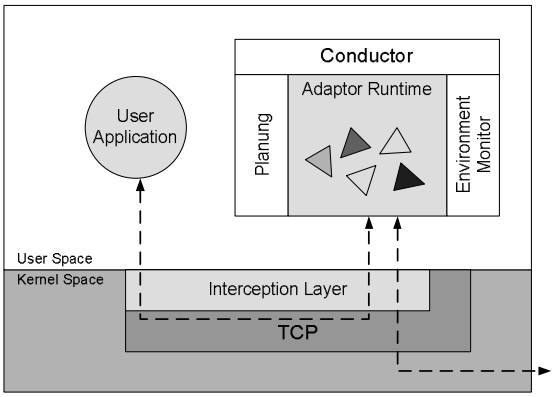


Abbildung 2-9: Architektur des Laufzeitsystems von Conductor-Knoten [YRE+01]

Conductor installiert für eine Verbindung zwischen einem Client und einem Server auf den Knoten entlang des Pfades Adaptoren entsprechend der verfügbaren Ressourcen und Netzwerktechnologien. Dazu werden TCP-Verbindungen während des Verbindungsaufbaues vom ersten Knoten mit Conductor Laufzeitsystem unterbrochen. Werden Daten bzw. Protokolle über die Verbindung übertragen, deren Adaption Conductor unterstützt, wird der Client mit dem ersten Conductor-Knoten verbunden. Der weitere Verbindungsaufbau zum Server wird von Conductor durchgeführt. Dazu werden Kennungspakete in Richtung des Servers gesendet, die von den nächsten Conductor-Knoten entlang des Pfades empfangen und bis zum Server weiter vermittelt werden. Auf diese Weise werden die Conductor-Knoten entlang des Datenpfades ermittelt. Zwischen den jeweils benachbarten Knoten wird dann eine direkte TCP-Verbindung aufgebaut. Eine Verbindung zwischen Client und Server besteht dementsprechend durch mehrere Teilverbindungen zwischen Conductor-Knoten (siehe Abbildung 2-10).

Conductor stellt eine Infrastruktur zur Planung von Sequenzen von Adaptoren bereit, die auf den Conductor-Knoten entlang des Pfades verteilt werden. In die Infrastruktur können verschiedene Planungsalgorithmen installiert werden, die prinzipiell Informationen über die verfügbaren Ressourcen auf dem Endgerät, auf den Knoten und zur Übertragung sowie Benutzerwünsche berücksichtigen können. Implementiert wurde ein einfacher Planungsmechanismus, der auf Templates basiert [YRE+01]. Dies sind vorab definierte Beschreibungen von Adaptorsequenzen und deren Verteilung. Templates sind in der Regel protokollabhängig und definieren die Adaptoren, die benötigten Ressourcen und Abhängigkeiten zwischen Adaptoren. Entsprechend des Planungsalgorithmus' wird der letzte Conductor-Knoten (evtl. der Knoten des Servers) Planungsknoten. Dieser erhält die gesammelten

Kennungsinformationen, die von den Conductor-Knoten während des Pfadaufbaus versendet werden. Die Kennungspakete der Knoten enthalten Informationen über die lokalen Verbindungen, die verfügbaren Ressourcen und Adaptoren der einzelnen Knoten. Unter Verwendung dieser Informationen werden die Adaptoren von Templates auf realen Conductor-Knoten verteilt. Der Plan wird vom Planungsknoten in Richtung Client entlang des Pfades versendet und von den Conductor-Knoten aufgebaut. Damit wird innerhalb eines Durchlaufs der gesamte Pfad erzeugt. Entsprechend kann der Pfad für die Kommunikation vom Server zum Client aufgebaut werden.

Conductor enthält im Gegensatz zu Ninja Path einen Mechanismus zur gesicherten Ende-zu-Ende Übertragung von Daten, der auch eine verlustbehaftete Adaption unterstützt. Entsprechend des Mechanismus' werden Daten anhand der Sequenznummern von TCP gekennzeichnet. Adaptoren können Datensequenzen zu logischen Segmenten zusammenfassen, die den ursprünglichen Bereich der Sequenznummern beibehalten. Logische Segmente können jedoch nicht wieder geteilt werden. Werden Daten innerhalb eines Segmentes verworfen, bleiben die Sequenznummern unverändert. Übertragungswiederholungen können sich damit ohne Kenntnis der durchgeführten Adaption auf die ursprünglichen Sequenznummern beziehen. Gehen Daten durch den Ausfall eines Adaptors oder Knotens verloren, kann der Empfänger feststellen, welches Segment zuletzt vollständig übertragen wurde. Da jeder Adaptor Sequenznummern auf logische Segmente abbilden kann, können Daten auf diese Weise eindeutig zur Übertragungswiederholung angefordert werden. Die Wiederholung beginnt dann beim darauf folgenden Segment.

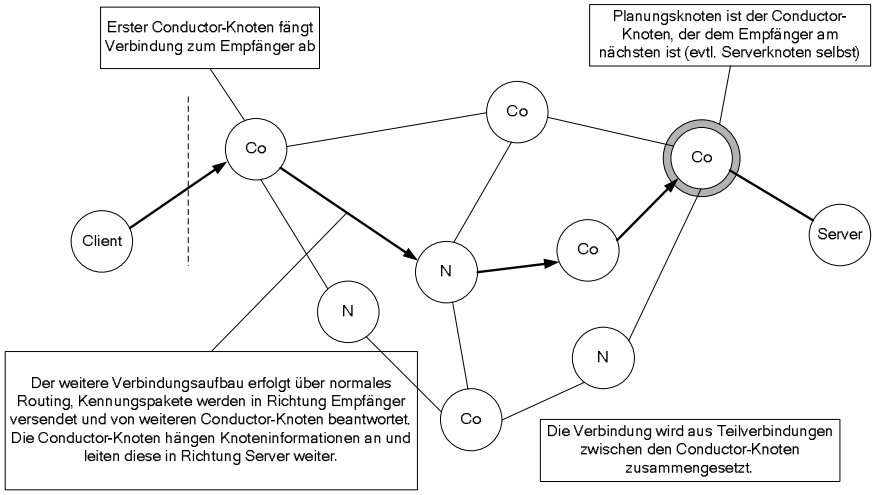


Abbildung 2-10: Erzeugung einer verteilten Adaptorsequenz in Conductor

Die Platzierung der Adaptoren kann bei Conductor auf allen Conductor-Knoten entlang eines Pfades erfolgen. Damit ist im Gegensatz zu Ninja eine wesentlich freiere Platzierung möglich, die auch Mechanismen auf den Endgeräten einschließt. Conductor nennt zwar die Möglichkeit der Einbeziehung von Umgebungsinformationen während der Planung, die konkret verwendeten Informationen und deren Anwendung werden jedoch nicht beschrieben. Ebenso werden Anpassungen an Änderungen der Ausführungsumgebung durch die Adaptoren selbst oder bei größeren Änderungen durch Anpassung der Adaptorsequenz erwähnt. Eine Beschreibung konkret angewendeter Regeln für die Adaption fehlt jedoch ähnlich wie bei Ninja. Die Kombination von Adaptoren erfolgt vergleichbar mit Ninja Path anhand der Ein- und Ausgabeprotokolle der Adaptoren. Conductor bezieht zusätzlich die benötigten Ressourcen der Adaptoren mit ein. Eine generische Beschreibung von Adaptorpfaden im Sinne einer Architekturbeschreibungssprache (siehe Abschnitt 2.5.4) existiert

jedoch auch in Conductor nicht. Ziel des Projektes war die Adaption von Diensten an heterogene Netzwerkverbindungen. Dazu wurde vor allem eine Laufzeitumgebung implementiert, die eine freie Platzierung von Adaptoren auf Knoten innerhalb der Netzwerkinfrastruktur einschließlich der Endgeräte ermöglicht.

2.2.2.3 Path Framework

Das Path-Framework [KiF00] verfolgt einen ähnlichen Ansatz wie Ninja Path und Conductor. Ein Pfad repräsentiert eine Sequenz verteilter Komponenten zur Datenadaption (Mediatoren), die eine ad hoc Kommunikation zwischen heterogenen und autonomen Komponenten in ubiquitären Infrastrukturen ermöglichen soll. Pfade werden dynamisch erzeugt und sollen vor allem Inkompatibilitäten zwischen Protokollen und Datenformaten ausgleichen. Die Infrastruktur besteht aus Mediatoren, Representatives und einem Koordinator (siehe Abbildung 2-11). Mediatoren dienen der Datentransformation und besitzen jeweils eine Ein- und Ausgabeschnittstelle für genau einen Datentyp. Zwei Mediatoren können dementsprechend kombiniert werden, wenn ihre Ein- bzw. Ausgabetypen übereinstimmen. Representatives sind die Gateways zwischen der Pfadinfrastruktur und Endpunkten der Kommunikation. Jeder Representative unterstützt genau ein Protokoll zur Kommunikation mit kompatiblen Endpunkten. Mediatoren und Representatives werden dem System durch das Anmelden beim Koordinator bekannt gemacht.

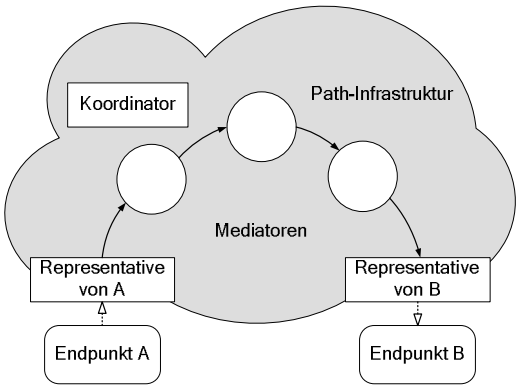


Abbildung 2-11: Architektur des Path Frameworks (nach [KiF00])

Um einen Pfad aufbauen zu können, muss für jeden Endpunkt der Kommunikation ein Representative mit kompatiblem Protokoll verfügbar sein. Außerdem muss eine Folge von Mediatoren existieren, welche die Daten des Senders in das Format des Empfängers transformieren kann. Ausgangspunkt der Pfaderzeugung ist ein logischer Pfad. Dieser muss zum größten Teil manuell erstellt werden. Unter der Voraussetzung, dass der logische Pfad aus der Typanforderung eines Endpunktes vollständig beschrieben wird, kann die Erzeugung aber automatisch erfolgen.

Die Implementierung des Frameworks basiert auf Operatoren und Konnektoren. Operatoren sind Codemodule mit einer steng getypten Ein- und Ausgabe. Die Schnittstellen sowie Informationen über die Instantiierung und Ausführung sind in einer XML-basierten Beschreibung enthalten. Operatoren dienen als Grundkomponenten für die Implementierung von Mediatoren, Representatives sowie weiterer Komponenten zur Datenverarbeitung (z. B. Aggregatoren und Disseminatoren, semantische Prozessoren). Konnektoren sind dagegen typneutral und implementieren die Datenübertragung über Netzwerkverbindungen. Sie enthalten eine Beschreibung der Charakteristiken des implementierten Protokolls (z. B. Zuverlässigkeit, Verzögerung, QoS, Sicherheit). Die Instantiierung des logischen

Pfades wird in drei Schritten realisiert. Im ersten Schritt wird allen Operatoren (Mediatoren und Representatives) ein Ausführungsort zugewiesen. Außerdem werden Konnektoren als Verbindungen zwischen verteilten Mediatoren definiert. Im zweiten Schritt werden die Beschreibungen der Teilpfade an die Ausführungsorte verteilt. Im dritten Schritt erfolgt dann die Instantiierung der Operatoren und Konnektoren innerhalb der verteilten Infrastruktur. Danach kann der Pfad ausgeführt werden.

Um die Automatisierung der Erzeugung logischer Pfade zu erhöhen, wurde das Typsystem in einigen Punkten erweitert. So können durch Wildcards und „Unknown“ Typen mit mehreren möglichen Werten bzw. ungebundene Typen spezifiziert werden. Attribute erlauben die Definition von Meta-Informationen für Typen in Form von Name-Wert Paaren. Außerdem wurden neben der Vererbung weitere Beziehungen zwischen Typen eingeführt (z. B. „enthält“ oder „repräsentiert“).

2.2.3 Protokolladaption auf Basis von Komponenten

Die in Abschnitt 2.1.2 beschriebenen Mechanismen zur Adaption von Kommunikationsprotokollen wurden fest in den Protokollcode integriert. Damit entstehen vorwiegend angepasste Protokolle, die bei bestimmten Problemstellungen eine verbesserte Leistungsfähigkeit gegenüber den Standardprotokollen bieten. Die nachfolgend vorgestellten Lösungen betrachten dagegen allgemeine Möglichkeiten zur Erweiterung und Adaption von Protokollen. Viele der Adaptionsmechanismen können in Form von Modulen kombiniert werden und damit die Funktionalität von Protokollen in generischer Weise erweitern. Zu dieser Klasse von Lösungen gehören auch programmierbare Netzwerke, die ähnliche Ziele verfolgen. Diese werden als Laufzeitmechanismen in Abschnitt 2.3.5 beschrieben.

2.2.3.1 Protocol Booster

Protocol Booster [FMS+98, MHM+98] ist ein Ansatz zur Adaption generischer Netzwerkprotokolle. Es soll einerseits eine schnellere Weiterentwicklung und Verbreitung von Protokollen und andererseits eine dynamische Anpassung von Protokollen an Netzwerkbedingungen ermöglicht werden. Protocol Boosters sind Hardware- oder Softwaremodule, die transparent in die Protokollverarbeitung eingefügt werden. Sie bestehen aus einem oder mehreren Elementen und können auf beliebigen Knoten innerhalb der Infrastruktur, einschließlich der Endgeräte, installiert werden. Protocol Booster arbeiten immer auf einem Basisprotokoll und können Nachrichten zu diesem Protokoll hinzufügen, entfernen oder verzögern. Damit bilden sie zwischen Booster-Paaren neue Protokolle. Booster sind jedoch niemals Quelle oder Empfänger eines Basisprotokolls und konvertieren auch keine Protokolle. Dadurch bleibt das Einfügen von Boostermodulen für die Kommunikationsendpunkte transparent. Insbesondere bleibt das Protokoll und damit eine mögliche Ende-zu-Ende Beziehung unverändert. Booster für das gleiche Basisprotokoll bilden gemeinsam mit diesem eine Protokollfamilie. Verschiedene Booster einer Familie können durch Aneinanderreihen oder Verschachteln kombiniert werden. Die Kombination von Boostern wird durch die Definition von Bedingungen (policies) für den Einsatz einzelner Booster und Meta-Bedingungen (meta policies) für die Kombination von Boostern gesteuert. Diese Definitionen bilden die Grundlage für die Entscheidung, wann ein Booster eingesetzt wird und welche Booster miteinander kombiniert werden können.

Protocol Booster können aus einem oder mehreren Elementen bestehen. Booster mit einem Element können das Basisprotokoll nicht verändern, insbesondere können sie keine zusätzlichen Nachrichten in den Datenstrom einfügen. Beispiele für Booster mit einem Element

sind die Berechnung der Prüfsumme in UDP-Paketen vor der Übertragung über Weitverkehrsnetze oder die Zusammenfassung von TCP-Bestätigungen für asymmetrische Verbindungen. Weitere Beispiele sind die Staubehandlung durch Erzeugen duplizierter Bestätigungen für TCP oder das Zwischenspeichern und Wiederholen unbestätigter Pakete ähnlich dem Snoop-Protokoll für TCP-Verbindungen zu mobilen Endgeräten. Booster aus zwei Elementen können zur Kommunikation untereinander Protokollnachrichten verändern und hinzufügen. Eine mögliche Anwendung ist die Jittersteuerung. Dabei versieht das erste Element alle Nachrichten mit einem Zeitstempel. Das zweite Element verzögert dann die Nachrichten entsprechend des Zeitstempels, um die Intervalle zwischen den Nachrichten auf Kosten einer höheren Antwortzeit wieder herzustellen. Weitere Beispiele sind die Implementierung negativer Bestätigungen oder das Einfügen einer Vorwärtsfehlerkorrektur transparent für das Basisprotokoll (siehe auch [MHM+98]).

Der Ansatz der Protocol Booster beschreibt eine Methode für einen modularen und inkrementellen Entwurf von Protokollen. Er zielt damit insbesondere auf die Infrastruktur programmierbarer Netzwerke und stellt eine Programmiermethode für diese dar. Durch die Modularisierung und Kombination ist eine leichte Anpassung, Erweiterbarkeit, Änderung und Weiterentwicklung von Protokollen möglich.

2.2.3.2 Transformer Tunnels

Transformer Tunnels [SuB01] beschreibt einen Ansatz, der ähnlich wie Protocol Booster einen kombinierten Einsatz verschiedener Mechanismen zur dynamischen Protokolladaption ermöglicht. Das grundlegende Konzept dafür sind Transformer Tunnel, die in ein Segment einer Datenverbindung eingefügt werden und durch die alle Daten der Verbindung fließen. Anfangs- und Endpunkte eines Tunnels können auf potentiell beliebigen Rechnern der Infrastruktur einschließlich der Endgeräte installiert werden. Zwischen den Endpunkten können sich ein oder mehrere Hops befinden. Pakete werden beim Eintritt in den Tunnel durch verschiedene Mechanismen an die Verbindungseigenschaften angepasst und danach zum Endpunkt der Verbindung übertragen. Am Endpunkt des Tunnels werden die ursprünglichen Pakete wieder hergestellt. Die Adaption erfolgt somit wie bei Protocol Booster transparent für Anwendungen und ohne Änderung der Protokolle an den Kommunikationsendpunkten.

Zu einem Tunnel können über eine Programmierschnittstelle dynamisch Transformationsfunktionen hinzugefügt oder aus diesem entfernt werden. Diese werden in der Reihenfolge ihrer Installation im Tunnel auf alle Pakete der Verbindung angewendet. Die transformierten Pakete enthalten alle notwendigen Informationen für die Rücktransformation am Endpunkt des Tunnels (z. B. die angewendeten Transformationen, Zieladresse, etc.). Innerhalb des Tunnels werden die Pakete über Punkt-zu-Punkt Verbindungen an den Endpunkt weiter vermittelt.

In der beschriebenen Lösung werden Tunnel für mobile Endgeräte mit Endpunkten auf dem Endgerät und auf der Basisstation aufgebaut. Auf der Basisstation wird ein Tunnel Manager installiert. Mobile Endgeräte können den Tunnel über diesen konfigurieren und bestimmte Transformationen anfordern. Mobile Endgeräte senden periodisch ein Signal an einen Tunnel Manager, solange sie den Tunnel aufrechterhalten möchten. Bei einem Wechsel der Basisstation wird der Tunnel Manager vom Endgerät benachrichtigt. Dieser initiiert danach die Übergabe des Tunnels an die neue Basisstation, auf der ein neuer Tunnel Manager aktiviert wird. Eine Umgehung des Tunnels ist für bestimmte Verbindungen eines Endgeräts möglich. Dazu werden die entsprechenden Verbindungsinformationen

beim Tunnel Manager registriert. Eintreffende Pakete der registrierten Verbindungen werden dann ohne Transformation vermittelt.

Der Ansatz stellt eine Programmierschnittstelle zur Implementierung von Transformationsfunktionen zur Verfügung. In [SuB01] werden verschiedene Transformationen beschrieben. Reassembly kombiniert im Rahmen einer gegebenen Maximum Transfer Unit (MTU) kleine Pakete zu größeren Paketen. Dazu werden ankommende Pakete, die kleiner als die MTU sind, eine bestimmte Zeitspanne verzögert. Treffen innerhalb dieser Zeitspanne weitere kleine Pakete ein, werden diese zusammengefasst. Energy Savings ist eine Transformation zum Einsparen von Energie durch das zeitweilige Abschalten des Netzwerkadapters auf dem Endgerät. Dieser wird in den Schlaf-Modus versetzt nachdem die Basisstation benachrichtigt wurde. Die Basisstation speichert eintreffende Pakete zwischen und benachrichtigt das Endgerät, das daraufhin wieder aktiv wird und die Pakete abruft. Buffering verwendet ebenfalls einen Puffer auf der Basisstation, um Paketverluste während Handover zu vermeiden. Encryption stellt verschiedene Verschlüsselungsverfahren zur Kommunikation im Tunnel zur Verfügung. Der Schlüsselaustausch ist nicht Teil der Transformation, sondern muss im Vorfeld stattfinden. Der Mechanismus Compression stellt eine verlustfreie Kompression der Pakete zur Verfügung. Weitere Transformationen können auf einfache Weise integriert werden.

Die Transformationen dieses Ansatzes entsprechen den Protocol Boostern mit zwei Elementen. Die Platzierung ist aber auf die Tunnelendpunkte beschränkt. Prinzipiell können aber mehrere Tunnel aneinander gereiht werden, um mehrere Segmente mit verschiedenen Verbindungstechnologien separat anzupassen. Auch Transformer Tunnel erhalten die Ende-zu-Ende Semantik der Verbindungen und arbeiten transparent für die Kommunikationsendpunkte.

2.2.4 Zusammenfassung

Die beschriebenen Lösungen untersuchen die Kombination von Mechanismen in Form von Komponenten. Die Ansätze aus Abschnitt 2.2.1 konzentrierten sich vor allem auf das Finden eines Pfades bzw. eines Graphen für die Erzeugung eines angeforderten Medienobjektes. Beide Konzepte betrachten eine Optimierung der gefundenen Lösung. Ninja Path, Conductor und das Path Framework betrachteten dagegen vor allem die Laufzeitmechanismen zur Instantiierung und Ausführung verteilter Pfade. Protocol Booster und Transformer Tunnel sind Lösungen speziell zur Adaption von Kommunikationsprotokollen.

Die Beschreibung der Komponenten erfolgt bei allen Lösungen anhand der Typen der ein- und ausgegebenen Medienobjekte. Diese charakterisieren die Funktion der Mechanismen und können zum automatisierten Aufbau von Pfaden bzw. Graphen verwendet werden. Diese Voraussetzung wird jedoch nicht von allen Adaptionsmechanismen erfüllt. Beispielsweise sind Caching und Queuing unabhängig von bestimmten Datentypen. Ebenso werden Mechanismen, die nicht das Format oder den Typ der Daten ändern, nicht durch die Ein- und Ausgabetypen charakterisiert. Die Lösungen bleiben deshalb auf eine Untermenge der Adaptionsmechanismen beschränkt, für die die oben genannte Voraussetzung erfüllt ist. Für die integrierte Betrachtung aller Adaptionsmechanismen sind jedoch erweiterte bzw. neue Ansätze erforderlich. Ebenso wird in allen Lösungen der Aufbau einer Sequenz von Komponenten untersucht, die einmal für ein bestimmtes Medienobjekt oder eine bestimmte Kommunikationsverbindung erfolgt. Änderungen dieser Sequenz in Reaktion auf Änderungen der Ausführungsumgebung wurden jedoch nicht betrachtet. Zur Kombination von Komponenten wird von allen Ansätzen das Muster der „Pipes und Filter“ zum Teil in erweiterter Form verwendet. Dieses stellt somit eine wesentliche Lösungsidee

für die lose und flexible Kopplung von Adaptionsmechanismen dar. Eine Beschreibung des Architekturmusters erfolgt deshalb in Abschnitt 2.5.1.

2.3 Architektur und Platzierung

In den Abschnitten 2.1 und 2.2 wurden einzelne Adaptionsmechanismen sowie deren Kombination zu Adaptionspfaden diskutiert. Die Platzierung der Mechanismen wurde in den vorgestellten Lösungen zum Teil betrachtet und stellt einen wesentlichen Gesichtspunkt für die Adaption in mobilen Infrastrukturen dar. So ist die Platzierung von Caching, Prefetching und der Serveremulation auf dem Endgerät die Voraussetzung für abgekoppelte Operationen. Ebenso müssen Daten vor der Übertragung zum mobilen Endgerät über eine drahtlose Verbindung angepasst werden, um die Datenmenge zu reduzieren. Nachfolgend werden deshalb allgemeine Ansätze zur Platzierung von Mechanismen untersucht. Der Ansatz des Stellvertreters und Middlewareplattformen stellen allgemeine Architekturen für verteilte Anwendungen dar. Bei diesen werden Komponenten bzw. Module fest platziert. Betriebssystemerweiterungen ermöglichen das Einfügen zusätzlicher Funktionalität insbesondere auf dem Endgerät. Die eingefügten Mechanismen arbeiten generisch und stehen den Anwendungen des lokalen Systems anwendungsübergreifend zur Verfügung. Eine dynamische Installation sowie die Änderung der Platzierung von Mechanismen zur Laufzeit werden durch mobile Codesysteme und programmierbare Netzwerke unterstützt. Damit kann dynamisch auf Änderungen der Ausführungsumgebung reagiert werden. In Abschnitt 2.3.5 wird außerdem ein Ansatz beschrieben, dessen Schwerpunkt auf der Verteilung von Adaptionsmechanismen in einer verteilten Infrastruktur liegt. Mit Hilfe dieser Lösung können Platzierungsentscheidungen unter Einbeziehung der Informationen über die aktuelle Ausführungsumgebung getroffen werden.

2.3.1 Der Ansatz des Stellvertreters

Eine grundlegende Lösungsidee stellt der Ansatz der indirekten Kommunikation über einen Stellvertreter dar. Dieser beinhaltet die Einführung einer Abstraktionsebene zwischen zwei miteinander kommunizierenden Komponenten, um deren direkte Interaktion zu entkoppeln (vergleiche z. B. [Sha86, FiG94, GHJ+01]). Die eingefügte Komponente agiert für die beiden interagierenden Komponenten als der jeweilige Kommunikationspartner. Entsprechend des Client/Server-Modells bietet die eingefügte Komponente dem Client die Schnittstelle des Servers an und agiert aus der Sicht des Servers als Client. Dadurch ist das Einfügen einer Stellvertreterkomponente und damit zusätzlicher Funktionalität transparent für existierende Anwendungen möglich.

In Mobile Computing Szenarien können drei grundlegende Möglichkeiten der Platzierung des Stellvertreters unterschieden werden. Eine Platzierung ist auf dem Clientrechner (1), auf einem Zwischenrechner im Festnetz (2) oder auf dem Serverrechner (3) möglich. Variante (1) unterstützt vor allem die Realisierung *abgekoppelter Operationen*, die es dem Benutzer z. B. durch Caching und Emulation der Serverfunktionalität ermöglichen, auch ohne eine bestehende Verbindung weiterzuarbeiten (siehe Abschnitt 2.1.1.1). Die Varianten (2) und (3) unterstützen ebenfalls die verbindungslose Arbeit, verlagern aber die Verarbeitung ins Festnetz und realisieren damit *autonome Operationen* (siehe Abschnitt 2.1.1.4). Der Stellvertreter in (2) und (3) repräsentiert den Benutzer außerdem permanent im Festnetz. Dadurch bleibt dieser auch während Phasen der Abkopplung erreichbar [SHB+98]. Weiterhin ergibt sich in den Varianten (2) und (3) die Möglichkeit der verbesserten Unterstützung drahtloser Verbindungen durch den Einsatz generischer Mechanis-

men zur Übertragung (z. B. Queuing oder angepasste Protokolle [SKS+99, Zen99]) sowie zur Reduzierung des Datenvolumens (z. B. verlustbehaftete Komprimierung, abhängig vom Datentyp [FoB96]). Außerdem können auch Funktionen auf Anwendungsebene, z. B. die Filterung von Anwendungsdaten nach Inhalt und Relevanz, ausgeführt werden. Die einzelnen Varianten können zu Architekturen mit mehreren Stellvertretern kombiniert werden.

2.3.2 Betriebssystemerweiterungen

Betriebssystemerweiterungen ermöglichen ein transparentes Einfügen zusätzlicher anwendungsübergreifender, d. h. anwendungsunabhängiger Funktionalität. So werden in CODA [Kis96] Systemaufrufe zum Cachemanager Venus umgeleitet. Dieser wurde als Kernelkomponente integriert und verarbeitet Dateiaufrufe von Anwendungen. Die Umleitung der Aufrufe ist transparent für Anwendungen, die weiterhin Standard-Betriebssystemaufrufe verwenden können, und trotzdem von der Abkopplungsunterstützung von CODA profitieren. Odyssey [NSN97] arbeitet auf ähnliche Weise. Auch bei diesem System werden Anwendungsaufrufe im Betriebssystemkern abgefangen und umgeleitet. In Conductor [YRE+01] werden Systemaufrufe umgeleitet, um TCP-Verbindungen transparent für die Endpunkte aufzutrennen und Adaptionskomponenten auf verschiedenen Knoten zwischen Sender und Empfänger einzufügen.

2.3.3 Mobile Codesysteme

Mobiler Code stellt kein neues Konzept dar. Die entfernte Abarbeitung von Batch-Jobs wurde z. B. bereits 1973 beschrieben [Bog73]. Weitere Beispiele sind die Prozessmigration in Betriebssystemen zur Lastverteilung [Thi91], [Nut96] und die Migration auf Objektebene [JLH+88, LJP93]. Die genannten Ansätze unterscheiden sich anhand der Granularität der bewegbaren Programmmodule und des Grades der Transparenz der Migration für den Programmierer. Mobile Codesysteme erweitern die genannten Ansätze vor allem durch die explizite Kenntnis von Ausführungsorten im System, wodurch die in klassischen verteilten Systemen anstrebte [CDK94] Ortstransparenz aufgelöst wird und die Steuerbarkeit von Migrationen durch den Programmierer bzw. die Programmkomponente selbst ermöglicht wird.

Für mobile Codesysteme können die folgenden Entwurfsparadigmen unterschieden werden (siehe [FPV98] sowie Abschnitt 2.3.3):

1. Remote evaluation: Entsprechend dieses Paradigmas wird Code entfernt auf dem Server ausgeführt. Der Client übergibt das auszuführende Codefragment sowie gegebenenfalls Initialisierungsparameter dem Server (code shipping). Auf diesem befinden sich die notwendigen Daten und Ressourcen zur Ausführung des Codes.
2. Code on demand: Nach diesem Paradigma wird Programmcode durch den Client beim Server angefordert und auf dem Client ausgeführt. Die notwendigen Daten und Ressourcen zur Ausführung des Codes befinden sich in diesem Fall auf dem Client.
3. Mobile Agenten: In diesem Paradigma befindet sich der auszuführende Code auf dem Client, einige der benötigten Ressourcen befinden sich jedoch auf einem anderen Rechner innerhalb des Netzes. Zur Ausführung migriert der Code gemeinsam mit zugehörigen Daten und dem Ausführungszustand zu dem Rechner mit den Ressourcen. Anders als bei den zuvor beschriebenen Mechanismen wird also nicht nur Code, sondern eine aktive Verarbeitungskomponente zwischen Rechnern übertragen.

Insbesondere mobile Agenten unterstützen die Ausführung autonomer Operationen auf beliebigen Rechnern innerhalb eines Netzes. Eine Voraussetzung ist jedoch die Verfügbarkeit der Ausführungsumgebung auf diesen Rechnern. Agenten können innerhalb dieser Ausführungsumgebung unabhängig vom Benutzer bzw. der Anwendung und damit einer Verbindung zum mobilen Endgerät arbeiten. Insbesondere können sie sich autonom bewegen, um z. B. lokal auf Ressourcen zuzugreifen oder auf Fehlersituationen zu reagieren. Ergebnisse dieser autonomen Verarbeitung können nach Aufbau einer Verbindung zur Anwendung auf dem Endgerät übertragen werden.

Mobiler Code

Mobile Codesysteme bestehen nach [FPV98] aus Ausführungsumgebungen auf den einzelnen Rechnern. Jede der Ausführungsumgebungen besitzt eine eindeutige Identität, über die auf den Rechner sowie die dort befindlichen Ressourcen explizit zugegriffen werden kann. Innerhalb einer Ausführungsumgebung befinden sich Ausführungseinheiten und Ressourcen (z. B. eine Datei innerhalb eines Dateisystems). Eine Ausführungseinheit repräsentiert einen sequentiellen Kontrollfluss (z. B. einen einfachen Prozess oder Thread). Sie besteht aus Programmcode (code segment) sowie Zustandsinformationen zusammengesetzt aus einem Datenbereich (data space) und einem Abarbeitungszustand (execution state). Der Datenbereich setzt sich aus einer Menge von Referenzen auf Ressourcen zusammen. Der Abarbeitungszustand enthält den Befehlszähler (instruction pointer), private Daten und den Aufrufstapel (call stack) der Ausführungseinheit.

Diese Komponenten können in mobilen Codesystemen zwischen Ausführungseinheiten bewegt werden. Entsprechend der übertragenen Bestandteile können zwei Arten von Mobilität unterschieden werden. „Starke“ Mobilität: ist die Fähigkeit, Code, Datenbereich und Abarbeitungszustand einer Ausführungseinheit zu übertragen. Bei „Schwacher“ Mobilität hingegen werden nur Code und Datenbereich sowie gegebenenfalls Informationen zur Initialisierung einer Ausführungseinheit übertragen. Schwache Mobilität erlaubt nicht die Übertragung des Abarbeitungszustandes.

Mobile Agenten

Mobile Agenten stellen nach [FPV98] neben remote evaluation (code shipping) und code on demand (code fetching) eine Form mobilen Codes dar. Remote evaluation ist die entfernte Ausführung von Code auf dem Server. Der Client übergibt das auszuführende Programm zusammen mit Parametern dem Server, der dieses abarbeitet. Dabei werden die Ressourcen des Servers genutzt. Code on demand bezeichnet die Abarbeitung von Code auf dem Client. Dieser fordert Code vom Server an und führt diesen dann lokal unter Verwendung eigener Ressourcen aus. Mobile Agenten können sich innerhalb von Ausführungsumgebungen (Orten) eines Agentensystems zwischen Rechnern bewegen (migrieren) und dadurch mit anderen Agenten und Diensten lokal kommunizieren. Anders als bei Remote Evaluation und Code on Demand stellen mobile Agenten aktive Einheiten dar, d. h, es wird nicht nur der Code selbst, sondern die gesamte Ausführungseinheit (Code, Daten, Abarbeitungszustand und Kontrollfluss) bewegt.

Agenten besitzen mit der Möglichkeit der lokalen Kommunikation und Datenverarbeitung durch die Fähigkeit der Migration ein hohes Potential zur Reduzierung der Datenmenge, die eine Anwendung über das Netzwerk übertragen muss. Aufgrund ihrer Autonomie und Mobilität können sie unabhängig von Netzwerkverbindungen und bestimmten Ressourcen arbeiten, wodurch die Flexibilität und Robustheit von Anwendungen erhöht wird (siehe auch [Joh98, LaO99]). Der Ansatz der mobilen Agenten bietet eine Vielzahl von Vorteilen,

die in ihrer Gesamtheit zu einem neuen Ansatz führen [HCK97]. Da jedoch jeder einzelne Vorteil auch durch eine alternative Technologie erreicht werden kann, existiert keine Anwendung, die ausschließlich mit mobilen Agenten realisiert werden kann. Aufgrund fehlender Sicherheitsmechanismen, insbesondere zum Schutz der Agenten vor der Ausführungsumgebung, ist der Einsatz mobiler Agenten auf Anwendungen mit geringen Sicherheitsanforderungen beschränkt [GBH98]. Dies verhinderte bisher eine breite Anwendung mobiler Agenten in realen Systemen.

Agentensysteme [PhK98, Whi94] stellen grundlegende Dienste wie Migration, Lokalisierung und Persistenz von mobilen Agenten zur Verfügung. Die heute verfügbaren Systeme (siehe z. B. [IKV00, LaC96, WPW97]) basieren größtenteils auf der Programmiersprache Java, die sich als de facto Standard etabliert hat. Diese unterstützt nur schwache Mobilität.

2.3.4 Programmierbare Netzwerke

Die Einführung neuer und weiterentwickelter Technologien und Protokolle in bestehende Netzwerkinfrastrukturen ist aufgrund der integrierten Struktur der Hardware und Software von Netzwerkkomponenten und der daraus resultierenden Notwendigkeit umfangreicher Standardisierungen sehr zeitaufwendig (z. B. RSVP [BZB+97], IPv6 [Wie02]). Gleichzeitig existieren Ansätze, die anwendungs- und nutzerspezifische Operationen im Netzwerk ausführen (z. B. verteilte hierarchische Caches, Web Proxies, Snoop auf Basisstationen, Firewalls, Mobile/Nomadic Filters).

Der Ansatz programmierbarer Netzwerke zielt auf diese Problemstellungen. Ziel ist die Entkopplung von Netzwerkdiensten von der Hardware der Netzwerkkomponenten und die Öffnung der Schnittstellen für Dritte. Dadurch soll die Programmierbarkeit und damit die Ausführung anwendungs- und nutzerspezifischer Operationen zur Datenübertragung auf den Netzwerkkomponenten ermöglicht werden. Dienste können dann auf einfache Weise und in kurzer Zeit integriert bzw. an veränderte Anforderungen angepasst werden. Außerdem steht eine standardisierte Plattform zur Verfügung, die eine Installation und Ausführung von Code auf Knoten innerhalb des Netzwerkes ermöglicht.

Der Ansatz erfordert die Erweiterung des Kommunikationsmodells von Netzwerken um ein Verarbeitungsmodell [CMK+99]. Eine mögliche Architektur wird in [CBZ+98] beschrieben. Diese wurde im Rahmen des DARPA Active Network Programms entwickelt. Die Funktionalität eines Netzwerkknotens wird demnach in Ausführungsumgebungen und das Betriebssystem für den Knoten unterteilt. Die Architektur unterstützt explizit mehrere Ausführungsumgebungen auf einem Knoten. Damit können verschiedene Protokolle parallel verarbeitet werden. Dies ermöglicht z. B. einen Vergleich von Ansätzen in einer Netzwerkumgebung und die parallele Verarbeitung herkömmlicher und aktiver Netzwerkprotokolle zur Wahrung der Abwärtskompatibilität.

Die Ausführungsumgebung implementiert die Funktionalität zur Installation und Ausführung von Programmcode. Sie bietet eine Menge von Schnittstellen zur Realisierung von Diensten und Protokollen. Das Betriebssystem des Knotens verwaltet dessen Ressourcen zur Kommunikation und Verarbeitung und steuert den Zugriff auf diese. Außerdem stellt es Funktionen für den Zugriff und die Manipulation des Knotenzustandes und grundlegende Sicherheitsdienste zur Verfügung. Auf diese Basisfunktionen kann über wohldefinierte Schnittstellen zugegriffen werden.

Das Knotenbetriebssystem implementiert Kanäle zur Kommunikation, über die Ausführungsumgebungen Pakete senden und empfangen können. Kanäle bestehen aus den physischen Verbindungen und dem Kommunikationsprotokoll. Die Zuordnung der Pakete zu

einem Kanal erfolgt über die Informationen im Nachrichtenkopf des Paketes. Pakete, die über physische Verbindungen auf dem Knoten eintreffen, werden durch diesen klassifiziert und einem Kanal zuordnet. Die Ausführungsumgebungen geben dazu bei der Erzeugung eines Kanals ein Muster an, das Pakete diesem Kanal eindeutig zuweist. Das Active Network Encapsulation Protokoll [ABG+97] bietet dazu eine Lösung. Der Protokollkopf enthält einen Identifikator, der eine eindeutige Zuordnung der Pakete zu einem Kanal ermöglicht.

Ein wesentliches Merkmal aktiver Netzwerke ist die Verfügbarkeit offener Schnittstellen, die eine Installation und Ausführung anwendungs- und nutzerspezifischen Programmcodes auf dem Knoten ermöglicht. Existierende Ansätze unterscheiden sich insbesondere durch den Grad der Programmierbarkeit der Netzwerkknoten. Dieser wird durch die Programmiersprache, die Art der Codeinstallation und dem Umfang der Schnittstellen bestimmt.

Anhand der Methode der Codeinstallation können zwei grundlegende Ansätze für programmierbare Netzwerke unterschieden werden [TSS+97, CMK+99]. Zum einen kann Code vorab und getrennt von der Datenübertragung installiert werden. In diesem Fall sind die aktiven Komponenten die Netzwerkknoten, die den Code zur Verarbeitung der Daten enthalten (Open Signaling). Zum anderen können die Pakete selbst aktiv sein, d. h. diese enthalten neben den Nutzerdaten auch den Programmcode, der zu ihrer Verarbeitung auf den Knoten ausgeführt werden soll [WGT98] (Active Networking). Kombinationen beider Ansätze werden beispielsweise in [AAH+98] diskutiert. Ein weiterer Aspekt ist die Granularität der Codeinstallation. Für aktive Pakete kann potentiell jedes Paket unterschiedlichen Code enthalten. Es kann aber auch für einen Strom oder für die gesamte Protokollverarbeitung Code installiert werden.

Die Programmiersprache wird vor allem durch ihren Sprachumfang und die Art der Ausführung charakterisiert [TeW96, CBZ+98]. Im einfachsten Fall können Parameter zur Selektion vordefinierten Codes eingesetzt werden. Spezialisierte Sprachen wie NetScript und PLAN [AAH+98] beschränken die Ausdrucksstärke der Programmiersprache, um die Korrektheit der Programme und Sicherheitsaspekte für die Ausführung durchzusetzen. Turing-vollständige Sprachen bieten die größte Flexibilität, stellen jedoch auch die höchsten Anforderungen in Bezug auf Sicherheit. Die Art der Ausführung beeinflusst vor allem die Portabilität der Programme und die Geschwindigkeit der Abarbeitung. Interpretierte Sprachen bieten die beste Portabilität, jedoch auf Kosten der Abarbeitungsgeschwindigkeit. Durch die Erzeugung und Abarbeitung von Zwischencode (z. B. Java Bytecode) oder die Übersetzung zur Laufzeit („On-the-Fly“) kann die Effizienz der Ausführung unter Wahrung der Portabilität erheblich gesteigert werden. Wird Maschinencode eingesetzt, können heterogene Knoten nur durch spezifische Versionen des Programms für die einzelnen Plattformen unterstützt werden.

Weitere Charakteristika von Ansätzen programmierbarer Netzwerke sind die zugrunde liegende Netzwerktechnologie, die verfügbaren Schnittstellen und die eingesetzten Sicherheitsmechanismen (siehe [CBZ+98, Pso99]). Von den definierten Schnittstellen abhängig sind die Möglichkeiten der Nutzung der Knotenressourcen. Insbesondere die Möglichkeit der Speicherung von und des Zugriffs auf Zustandsinformationen wirkt sich auf den Protokollentwurf aus.

Durch die flexible Programmierbarkeit von Netzwerkknoten bieten programmierbare Netzwerke eine Vielzahl von Möglichkeiten für die Adaption von Anwendungen. Insbesondere die flexible Platzierbarkeit von Programmcode kann ausgenutzt werden, um Adaptionsmechanismen auf Netzknoten zu verteilen, auf denen diese am effizientesten eingesetzt werden können.

2.3.5 Automatische Platzierung

Active Pipes [KRW+00, KRW+01] beschreibt eine Lösung zur optimalen Verteilung von Anwendungsoperationen innerhalb einer Netzwerkinfrastruktur basierend auf dem Ansatz programmierbarer Netzwerke. Basis des Ansatzes ist das Active Pipe Paradigma. Dieses beschreibt Ende-zu-Ende Verbindungen als eine Folge von Operationen, die auf dem zu übertragenden Datenstrom ausgeführt werden. Jede Operation entspricht dabei einem Codemodul, das entlang des Datenpfades auf einem Knoten innerhalb der Netzwerkinfrastruktur installiert werden muss. Der Ansatz schließt Endgeräte als mögliche Orte der Platzierung ein.

Active Pipes unterscheidet zwei Arten von Modulen:

1. Erforderliche Module sind für die korrekte Ausführung der Anwendung notwendig und müssen genau einmal abgearbeitet werden.
2. Optionale Module verbessern die Qualität der Anwendung, müssen aber nicht unbedingt ausgeführt werden. Optionale Module verändern in der Regel das Datenformat nicht und können gar nicht, einmal oder mehrmals abgearbeitet werden.

Für diese Codemodule können Eigenschaften und Restriktionen für die Platzierung in Form von Attributen mit Name-Wert Paaren definiert werden. Dabei werden drei Typen von Attributen unterschieden:

1. Netzwerkattribute werden allen Knoten und Netzwerkverbindungen der Infrastruktur zugewiesen. Sie definieren statische und dynamische Zustandsinformationen. Statische Attribute sind z. B. Netzwerkadresse, Domäne, Standardgateway, usw. Dynamische Attribute sind z. B. aktuelle Last, verfügbare Bandbreite, usw.
2. Selektoren werden Active Pipe Definitionen zugewiesen. Sie beschränken die Menge von Knoten und Netzwerkverbindungen, auf denen Codemodule installiert werden können (z. B. Installation aller Codemodule in einer bestimmten Domäne oder einem bestimmten IP-Adressbereich).
3. Installationsbedingungen werden Codemodulen zugewiesen. Sie definieren Bedingungen für die Installation von Codemodulen, die Verbindungen und Knoten erfüllen müssen. Zunächst wird bei der Auswertung der Attribute die Menge möglicher Pfade durch Selektoren eingeschränkt. Anschließend wird die Platzierung der Codemodule durch Vergleich der Installationsbedingungen mit den Netzwerkattributen der einzelnen Knoten und Verbindungen verglichen.

Active Pipes können einschließlich der beschriebenen Attribute im Rahmen der Active Pipe Grammatik beschrieben werden. Die Grammatik beschreibt einen logischen Pfad, der auf das physische Netzwerk abgebildet werden muss. Ergebnis der Abbildung ist die Festlegung der Knoten für die Platzierung der einzelnen Codemodule und des Pfades, über den der Datenstrom durch das Netzwerk übertragen wird. In [KRW+01] wird ein Algorithmus beschrieben, der eine Active Pipe auf einen Pfad durch die Infrastruktur mit minimalen Verarbeitungs- und Übertragungskosten abbildet. Der Algorithmus beschreibt die Infrastruktur als gerichteten Graphen $G=(V, E)$ mit der Menge V aller Knoten und Menge E aller Kanten im Netzwerk. Jeder Verbindung $e \in E$ werden Übertragungskosten $c(e)$, jedem Knoten $v \in V$ Verarbeitungskosten $c(v)$ zugewiesen. Der Quellknoten ist mit s, der Zielknoten mit d gekennzeichnet. Die Knotenmenge $r \in R$ mit $R \subseteq V$ enthält alle Knoten, welche die Anforderungen der Verarbeitungskomponenten erfüllen. Da die Suche nach dem kürzesten Pfad mit mehreren Kostenmetriken NP-vollständig ist, werden die Verarbeitungsknoten in den Wertebereich der Übertragungsknoten transformiert. Die Kosten des

gesamten Pfades sind damit die Summe der Übertragungskosten und der Verarbeitungskosten auf den Knoten.

Zur Lösung des Problems durch die Suche des kürzesten Pfades wird der Graph durch Kopieren transformiert. Für eine Active Pipe mit k Operationen wird ein Graph mit k+1 Ebenen $G_1=(V_1, E_1)$ bis $G_{k+1}=(V_{k+1}, E_{k+1})$ erzeugt. Für jeden Knoten v des Ursprungsgraphen existieren dann die Knoten v_1 bis v_{k+1} in den entsprechenden Ebenen. Quellknoten wird der Knoten s_1, Zielknoten der Knoten d_{k+1}. Die Verarbeitungskosten werden durch zusätzliche Kanten zwischen den Ebenen abgebildet (siehe Abbildung 2-12). Eine Beschränkung der möglichen Platzierung einer Operation o_i auf eine Knotenmenge R wird durch die Definition entsprechender Kanten zwischen den Ebenen i und i+1 erreicht. Dabei wird für jeden Knoten $r \in R$ zwischen den Ebenen i und i+1 eine Kante (r_i, r_{i+1}) mit den transformierten Verarbeitungskosten $c_i(r)$ erzeugt. Die Lösung kann durch Suche des kürzesten Pfades durch diesen Graphen gefunden werden. Die Operationen werden auf den Knoten platziert, an denen ein Übergang auf die nächste Ebene des Graphen erfolgt. Optionale Module können ebenfalls entsprechend einer gegebenen Knotenmenge R auf allen Knoten auf dem Pfad installiert werden. Eine entsprechende Graphentransformation erfolgt dabei in einer Ebene. Dabei werden für jeden Knoten $r \in R$ die Gewichtungen aller abgehenden Kanten um die Verarbeitungskosten auf dem Knoten r erhöht. Die Suche nach dem kürzesten Pfad bezieht so die Verarbeitung der bedingten Operation auf den entsprechenden Knoten mit ein. Für alternative Operationen werden separat optimale Pfade ermittelt. Die Alternative, die den Pfad mit geringsten Kosten ergibt, wird ausgewählt (siehe auch [KRW+00, CTW01]).

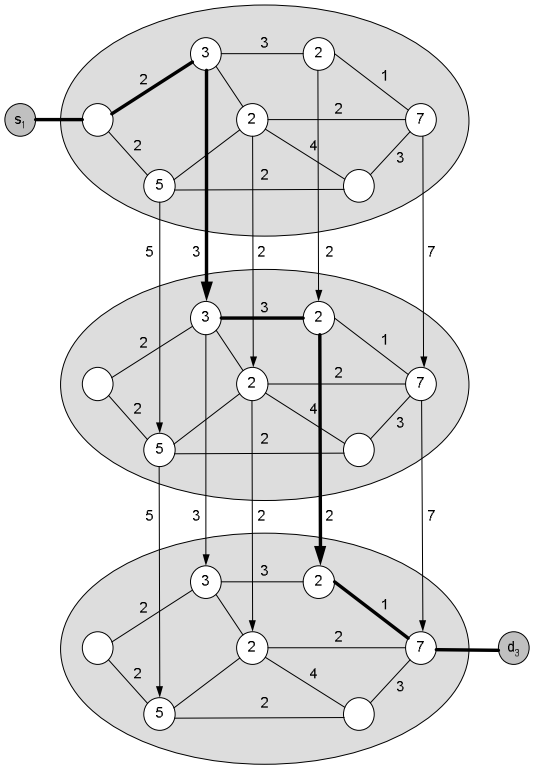


Abbildung 2-12: Graphtransformation für k=2 erforderliche Operationen

Die beschriebene Lösung setzt die Kenntnis der gesamten Infrastruktur voraus. In größeren Netzen sind diese Informationen in der Regel nicht verfügbar. Eine Erweiterung des Algorithmus' arbeitet auf Basis einer Hierarchie von Knotengruppen (peer groups). Eine Knotengruppe erscheint als ein Knoten auf der nächst höheren Hierarchieebene. Jede Knoten-

gruppe besitzt einen Hauptknoten (peer group leader), der die Knotengruppe repräsentiert und die aggregierten Knoteninformationen enthält. Durch die Aggregation der Knoteninformationen können diese an die nächst höhere Ebene weitergegeben werden. Aggregationen werden durch unterschiedliche Verknüpfungen der Informationen entsprechend ihrer Semantik gebildet. Der beschriebene Algorithmus kann dann ausgehend von der höchsten Hierarchieebene absteigend angewendet werden, um einen Pfad aufzubauen. Der ermittelte Pfad stellt jedoch kein globales Optimum dar.

2.3.6 Zusammenfassung

Die Platzierung von Komponenten ist für die Adaption aus mehreren Perspektiven von Bedeutung. So bestimmt die Platzierung, auf welchem Rechner für die Ausführung einer Komponente Ressourcen benötigt werden. Die Anzahl der Komponenten, die auf mobilen Rechnern platziert werden kann, wird durch die lokal verfügbaren Ressourcen begrenzt. Damit sind abgekoppelten Operationen, für die Komponenten für Caching, Prefetching und eine lokale Serveremulation notwendig sind, nur auf Geräten mit ausreichend Speicher und Rechenleistung sinnvoll möglich.

Zur Unterstützung schmalbandiger Kommunikationskanäle ist eine Adaption der Datenmenge vor der Übertragung entscheidend. Entsprechende Komponenten müssen deshalb in der Regel auf Rechnern im Festnetz platziert werden. Dabei spielt meist weniger die Ressourcenverfügbarkeit eine Rolle, vielmehr müssen die Komponenten in existierende Anwendungen eingefügt werden. Während Betriebssystemerweiterungen dies lokal ermöglichen und hauptsächlich auf dem Endgerät angewendet werden, stellt der Ansatz des Stellvertreters einen allgemeinen Lösungsansatz dar. Beide Konzepte unterstützen aber die transparente Integration von Adaptionsmechanismen in Anwendungen. Betriebssystemerweiterungen bleiben dabei auf Mechanismen beschränkt, die anwendungsübergreifend arbeiten. Der Stellvertreteransatz unterstützt dagegen das Einfügen sowohl generischer als auch anwendungsspezifischer Mechanismen. Drei- und mehrschichtige Architekturen, wie sie beispielsweise von Application Servern unterstützt werden, bieten für die Verteilung eine standardisierte Plattform und unterstützen insbesondere auch die Implementierung des Stellvertreteransatzes.

Neben der statischen Platzierung existiert mit mobilem Code auch die Möglichkeit einer dynamischen Installation und einer Änderung der Platzierung zur Laufzeit. Mit mobilen Agenten und programmierbaren Netzwerken stehen dafür zwei Plattformen für unterschiedliche Anwendungsbereiche zur Verfügung. Mobile Agenten eignen sich für die Realisierung vollständiger Anwendungen, während programmierbare Netzwerke auf die Erweiterbarkeit und Flexibilität der Netzwerkkommunikation zielen. Die Migration von Code und damit eine dynamische Änderung der Platzierung ist mit beiden Technologien möglich und mit entsprechenden Kosten verbunden. Diese müssen insbesondere gegen die Kosten einer entfernten Kommunikation abgewogen werden, denn nicht in jedem Fall ist eine Migration weniger aufwendig als eine entfernte Kommunikation (siehe dazu [SSZ01]). Ebenso ist eine zu feingranulare Aufteilung und Verteilung von Anwendungen zu vermeiden, da sich mit der Anzahl der Komponenten auch der Kommunikationsaufwand erhöht, insbesondere wenn die Komponenten verteilt werden (siehe [SHZ+99]).

Generell können somit Anwendungs- bzw. Adaptionskomponenten auf dem Endgerät, Rechnern im Festnetz sowie auf dem Anwendungsserver platziert werden. Durch mobilen Code wird zum einen eine dynamische Installation und zum anderen eine dynamische Änderung der Platzierung von Komponenten möglich.

2.4 Kontext zur Steuerung von Adaptionsprozessen

Anwendungen werden innerhalb einer Ausführungsumgebung abgearbeitet, die bestimmte Eigenschaften aufweist. Mobile Ausführungsumgebungen sind sehr heterogen und unterliegen einer hohen Dynamik. Die Kenntnis der Eigenschaften der Ausführungsumgebung, insbesondere der verfügbaren Ressourcen, ist deshalb für die Adaption von entscheidender Bedeutung.

Entsprechend der Definition von Kontext nach [Dey01] (siehe Abschnitt 1.4) kann die Gesamtmenge der Informationen über alle Entities als Kontext bezeichnet werden, die für eine Interaktion zwischen Benutzer und Anwendung relevant sind (einschließlich des Benutzers und der Anwendung selbst). Zur Adaption von Anwendungen in mobilen Infrastrukturen sind gemäß den Anforderungen A1.1 bis A1.4 aus Kapitel 1 Informationen über Endgeräte, Netzwerkverbindungen, die Anwendungssituation und den Benutzer von besonderem Interesse.

Im Allgemeinen ist Anwendungen nur eine geringe Teilmenge dieses Kontextes explizit bekannt. Das Erfassen und zur Verfügung stellen aller weiteren impliziten Informationen ist eines der Ziele der Forschung im Bereich Context-Awareness. Dadurch können implizite Informationen auf die gleiche Weise für die Adaption genutzt werden, wie explizit verfügbare Informationen. Kontext stellt somit eine wesentliche Quelle von Informationen zur Steuerung der Adaption dar.

2.4.1 Verfügbare Informationen

Kontext wird nach [Dey01] aus der Sicht eines Entities definiert, dessen Situation charakterisiert werden soll. Ein Entity kann dabei jede Person, jeder Ort oder jedes Objekt sein, das relevant für eine Interaktion bzw. Verarbeitung ist. Damit gehört eine Vielfalt an Informationen, insbesondere auch Informationen über mehrere verschiedene Entities zu Kontext. Neben der Person und dem Ort können auch Anwendungen, Dienste und Geräte Kontext liefern. Informationen können physikalischer Natur (z. B. ein geographischer Ort, die Zeit oder die Temperatur) oder technischer Natur sein (z. B. Display und Speichergröße eines Gerätes, Datenrate und Verzögerungszeit eines Kommunikationskanals).

Kontextklasse	**Beispiele**
physikalischer Kontext	Ort, Zeit, Temperatur, Lichtintensität
technischer Kontext	Display- und Speichergröße des Endgerätes, Datenrate und Verzögerung des genutzten Kommunikationskanals, verfügbare Energie
persönlicher Kontext	Adresse, Telefonnummer, Termine, bekannte Personen, Voreinstellungen für Anwendungen
situationsbezogener Kontext	Aktivität, Rolle und Aufgabe des Benutzers

Tabelle 2-2: Einteilung von Kontext in vier Klassen

Neben diesen in der Regel durch Sensoren oder Messungen erfassbaren bzw. bekannten oder explizit verfügbaren Informationen können auch Informationen über Personen (z. B. Adresse, Telefonnummer, Termine, bevorzugte Art der Bezahlung, Voreinstellung für Anwendungen) und die Anwendungssituation (z. B. Rolle des Benutzers, Aktivität und Aufgabe) zur Charakterisierung der Situation eines Entities genutzt werden. Entsprechend dieser Überlegungen kann Kontext in die folgenden vier Klassen eingeteilt werden, die in Tabelle 2-2 dargestellt werden (siehe [Gos01]).

Erfassung von Kontext

Wesentlich für die Verfügbarkeit und Nutzbarkeit von Kontext ist die Möglichkeit der Erfassung der Informationen. Vor einer Nutzung von Kontext müssen die entsprechenden Informationen erfasst und in einer für die Weiterverarbeitung durch Rechner geeigneten Weise zur Verfügung gestellt werden. Hinsichtlich der Erfassung können zwei Arten von Kontext unterschieden werden: Basiskontext und abgeleiteter Kontext (siehe auch [Lös02]).

Basiskontext wird durch ein geeignetes Verfahren direkt aus der Umgebung ermittelt. Dabei können zwei grundlegende Methoden unterschieden werden:

- Manuelle Ermittlung: Die ist die einfachste Methode zur Kontextgewinnung. Informationen werden über eine Benutzerschnittstelle direkt vom Benutzer abgefragt. Die Grenze zu gewöhnlichen Benutzereingaben ist fließend.
- Automatische Ermittlung: Zur automatischen Gewinnung von Kontext stehen abhängig von der Kontextart verschiedene Methoden zur Verfügung. So kann physikalischer Kontext durch Sensoren gemessen werden [SBG99, GrS01]. Informationen über den Benutzer können durch die Beobachtung seines Verhaltens ermittelt werden [JöM99]. Außerdem können Informationen von Anwendungen bzw. Informationsspeichern (z. B. Kalenderanwendung, Daten- und Wissensbank) abgefragt werden.

Abgeleiteter Kontext wird durch Abstraktion und Verknüpfung aus Basiskontext erzeugt. Insbesondere kann auf diese Weise aus durch Sensoren ermittelten Rohdaten ein von Anwendungen verarbeitbarer Wert gebildet werden. Außerdem können durch die Aggregation von Kontextwerten und die Anwendung von Wissen und Regeln höherwertiger Kontext abgeleitet werden (z. B. eine Wetterinformation aus Basiswerten wie Temperatur und Luftdruck oder eine Anwendungssituation aus der Uhrzeit, dem Aufenthaltsort und den dort befindlichen Personen). Diese Konzepte werden im Context Toolkit [Dey01] durch die Abstraktionen Interpretatoren und Aggregatoren unterstützt. Untersucht wurde abgeleiteter Kontext z. B. in [SAT+99] für die Ermittlung des Benutzungskontextes eines PDAs. Dabei wird aus Basiskontext (Beschleunigung, Lichtintensität, Umgebungsgeräusche, usw.) auf höherwertige Informationen (z. B. „Gerät befindet sich im Koffer“) geschlossen.

2.4.2 Darstellung von Kontext

Kontext besitzt einige grundlegende Eigenschaften, die Gewinnung, Darstellung und Nutzung beeinflussen. So werden mit den Prinzipien Lokalität und Zeitnähe in [ScG01] zwei wesentliche Eigenschaften von Kontext identifiziert und modelliert. Kontext besitzt demnach einen Ursprungsort und eine Ursprungszeit, von denen die Relevanz jedes Kontextwertes abnimmt. So ist die Relevanz eines Wertes am Ursprungsort bzw. zur Ursprungszeit maximal und nimmt mit wachsendem Abstand ab. Ab einem bestimmten Abstand von einem Bezugspunkt hat ein Kontextwert keine Relevanz mehr. Durch diese Eigenschaften ergeben sich weitere Aspekte für die Verteilung und Gültigkeit von Informationen. Durch die begrenzte Gültigkeit von Kontext ist die Verteilung räumlich und die Lebensdauer zeitlich begrenzt. Eine Verteilung ergibt sich dabei in einem verteilten System auf natürliche Weise. Insbesondere ist damit keine Notwendigkeit für eine zentrale Komponente gegeben.

Unter Berücksichtigung der genannten Eigenschaften wird Kontext in [ScG01] durch ein 6-Tupel K=(ID, Beschreibung, Einheit, Wertebereich, Wert, Wahrscheinlichkeit) beschrieben. Der Zugriff erfolgt durch die Definition von Schablonen KS=(ID|*, Beschreibungs-

schablone|*, Einheit|*, minimale Wahrscheinlichkeit|*). Das Sternsymbol „*“ beschreibt dabei beliebige Werte. Eine Anfrage liefert entsprechend ihrer Schablone alle Werte mit der angegebenen ID, einer passenden Beschreibung, der entsprechenden Einheit und einer Wahrscheinlichkeit, die größer als die in der Schablone gegebene Wahrscheinlichkeit sein muss. Anfragen erfolgen immer ausgehend von einem Bezugspunkt hinsichtlich Zeit und Ort. Die Plattform bildet intern die zeitliche und örtliche Relevanz von Kontext auf beliebige monoton fallende Funktionen ab, die einem Kontextlieferanten zugewiesen werden. Mit Hilfe dieser Informationen können Ergebnisse für Anfragen ausgehend von bestimmten Bezugspunkten ermittelt werden.

2.4.3 Kontextnutzung

Nach [ScG01] ist ein System mit Kontextbezug ein System, das die Fähigkeit hat, Aspekte der Umgebung als Kontext zu erfassen und diese für ein kontextbezogenes Verhalten zu nutzen. Ein kontextabhängiges System weist also zwei wesentliche Eigenschaften auf. Es besitzt die Fähigkeit, Kontext wahrzunehmen und es besteht ein definierter Zusammenhang zwischen den wahrgenommenen Informationen und dem Verhalten des Systems.

Allgemein ist eine Verarbeitung also *kontextabhängig*, wenn sie die Fähigkeit zur Wahrnehmung von Kontext besitzt und ein definierter Zusammenhang zwischen Kontextwerten und Verarbeitungsergebnissen besteht, d. h. sich in Reaktion auf Änderungen des Kontextes die Verarbeitungsergebnisse in definierter Weise ändern.

Kontextabhängige Anwendungen, die als Prototypen im Forschungsbereich „Context-Awareness“ entstanden, verwenden vorwiegend personenbezogene und insbesondere nicht-technische Informationen. Sie zielen auf die Verbesserung der Interaktionen zwischen Mensch und Anwendung. Der meistgenutzte Kontexttyp ist der Ortskontext, d. h. der Aufenthaltsort. Ortsinformationen spielen insbesondere in mobilen Systemen eine wichtige Rolle, da sich durch die Mobilität der Ort und damit die Systemumgebung des Benutzers häufig ändern. Ortsinformationen können außerdem durch verschiedene Technologien ermittelt werden und stehen in zellenbasierten Mobilfunksystemen inhärent zur Verfügung. Weiterhin lassen sich auf der Basis von Ortsinformationen im Allgemeinen Rückschlüsse auf weiteren Kontext ziehen.

Ortsinformationen werden beispielsweise im Active Badge System [WHF+92], einer Büroanwendung, verwendet. In diesem System werden Aufenthaltsorte von Personen in einem Gebäude ermittelt und benutzt, um deren Erreichbarkeit zu erhöhen. In ähnlicher Weise werden in Touristenführern und Fieldwork-Anwendungen Ortsinformationen verwendet. So werden dem Benutzer des GUIDE-Systems der Universität Lancaster [CDM+00] oder dem Cyberguide [AAH+97] abhängig vom Standort Informationen über Sehenswürdigkeiten angezeigt. Fieldwork-Anwendungen wie [PRM98] nutzen Ortsinformationen zur Annotation weiterer im Gelände erfasster Daten bzw. zur Auswahl von Informationen für den Benutzer. Weitere wichtige Kontexttypen sind die Identität, der Zeitpunkt und die Aktivität [Dey00]. Diese Informationen charakterisieren in markanter Weise die Situation von Entities (siehe auch [MDA00]).

2.4.3.1 Zugriff auf Kontext

In [Dey00] werden grundlegene Aktionen identifiziert, die Anwendungen als kontextabhängig charakterisieren. Diese drei grundlegenden Aktionen sind:

1. Kontextabhängige Präsentation von Informationen bzw. Diensten,
2. Automatische Ausführung von Diensten (Triggering durch Kontext) und
3. Assoziation von Informationen mit Kontext für eine spätere Suche.

Demnach können zum einen Informationen präsentiert und zum anderen kontextabhängige Aktionen ausgeführt werden. Dabei kann Kontext als reine Information zur Charakterisierung des Zustandes eines Entities oder als Ereignis der Änderung des Zustandes verwendet werden. Diese beiden orthogonalen Kategorien können zur Einordnung von Zugriffen auf Kontext durch Anwendungen verwendet werden. Dies wird in Tabelle 2-3 dargestellt.

	Zustand	**Ereignis**
Information	Aktuellen Standort des Benutzers Anzeigen oder mit Daten assoziieren, z. B. eine ortsabhängige Liste verfügbarer Drucker [SAW94]	Information „Person X betritt den Raum“ anzeigen oder mit Daten assoziieren
Aktion	Ein Dokument auf dem nächstgelegenen Drucker ausdrucken	Bei einem Verbindungsabbruch eine lokale Verarbeitung von Daten aktivieren

Tabelle 2-3: Grundlegene Aktionen kontextabhängiger Anwendungen (nach [Dey00])

Somit existieren zwei grundlegende Arten des Kontextzugriffs (siehe Abbildung 2-13):

1. *Aktive Kontextnutzung*: Kontextwerte werden explizit vom Kontextdienst abgerufen (Pull-Prinzip). Mit diesem Prinzip kann sowohl auf aktuellen als auch auf historischen Kontext zugegriffen werden. Welche Kontextwerte abgerufen werden, wird durch die Anfrage des Kontextbenutzers spezifiziert. Auslöser einer Anfrage ist immer der Kontextbenutzer, der zu einem bestimmten Zeitpunkt Kontextwerte benötigt.
2. *Passive Kontextnutzung*: Der Kontextdienst sendet Kontextwerte in Form von Ereignissen an einen Kontextbenutzer, wenn sich Kontextwerte ändern (Push-Prinzip). Der Kontextnutzer abonniert bei diesem Abfrageprinzip Ereignisse über die Änderung bestimmter Kontextwerte und wartet dann passiv auf das Eintreffen von Änderungsnachrichten. Tritt eine Änderung ein, erzeugt der Kontextdienst ein entsprechendes Ereignis und sendet dieses an alle Kontextbenutzer, die dieses Ereignis abonniert haben.

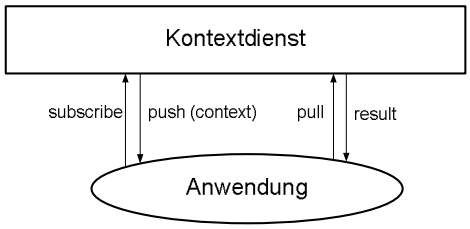


Abbildung 2-13: Zugriffsmethoden auf einen Kontextdienst

Beide Arten der Kontextnutzung sind zur Steuerung der Adaption relevant. Kontext wird aktiv genutzt, wenn Verarbeitungsoperationen ausgeführt werden, die kontextabhängig arbeiten. Eine passive Kontextnutzung erfolgt, wenn Adaptionsoperationen durch das Auftreten von Ereignissen aktiviert werden.

2.4.3.2 Adaption mit explizitem Kontextzugriff

Wie erläutert, greifen adaptive Anwendungen auf Informationen über die Ausführungsumgebung zu und nutzen damit implizit Kontext. Nachfolgend sollen ausgewählte Beispiele vorgestellt werden, die Umgebungsinformationen im Sinne von Kontext zur Steuerung der Adaption verwenden.

In [SuB01] wird ein Framework vorgestellt, das Verbindungsinformationen auf dem Endgerät zur Verfügung stellt und es Protokollen unterschiedlicher Schichten ermöglicht, auf diese Informationen zuzugreifen und sich entsprechend anzupassen. Durch Erweiterungen des Betriebssystemkernels um eine Programmierschnittstelle können unterschiedliche Quellen, wie Gerätetreiber und Protokolle, Informationen über das Netzwerk an das Framework übergeben werden. Das Framework definiert über den Wertebereich jedes verfügbaren Wertes eine Menge von Markierungswerten (watermarks). Diese Markierungswerte definieren Schwellwerte, die entweder ein Minimum oder ein Maximum darstellen. Sinkt ein Wert unter ein definiertes Minimum, erzeugt das Framework ein entsprechendes Ereignis. Gleiches gilt für die Überschreitung eines Maximums. Umfasst die Änderung mehrere Minima bzw. Maxima, werden mehrere Ereignisse erzeugt. Durch die Definition der Schwellwerte wird ein ständiges Generieren von Ereignissen verhindert, wenn ein Wert im Bereich eines bestimmten Schwellwertes schwankt. Insbesondere wird damit eine alternierende Adaption in Grenzbereichen verhindert.

Die Ereignisse werden Anwendungen und Kommunikationsprotokollen in Form von ICMP-Nachrichten zugänglich. Eine Erweiterung der Socketimplementierung ermöglicht die Auswertung von Ereignissen und eine Adaption von Protokollen auf der Transportschicht. Anwendungen können über die erweiterte Socketschnittstelle für bestimmte Ereignisse Adaptionsmechanismen registrieren. Zum Auslösen der Mechanismen sieht das Framework die Deklaration von Bedingungen vor, die aus einfachen Ereignissen oder einer logischen Verknüpfung von Ereignissen bestehen können. Betrachtete Adaptionsmechanismen sind für UDP die optionale Berechnung von Checksummen und ein Schlafzustand, in dem Pakete zwischengespeichert und verzögert versendet werden. In TCP kann die Größe des initialen Sendefensters und der Timeout-Wert des Retransmission Timers adaptiert werden. Auf Anwendungsebene werden das Verzögern der Datenübertragung bei einer Verbindung mit hoher Fehlerwahrscheinlichkeit für eine FTP-Anwendung [BIP+99] und die Adaption von Anwendungsdaten an die aktuelle Netzwerkverbindung genannt [FGC+98].

Der Ansatz stellt Informationen über eine Teilmenge des Kontextes mobiler Infrastrukturen zur Verfügung. Diese können von Anwendungen zur Adaption genutzt werden. Das Framework ermöglicht insbesondere die explizite Beschreibung der Bedingungen für die Aktivierung von Adaptionsmechanismen. Dazu werden Ereignisse auf generische Weise durch logische Operationen (AND, OR, NOT) miteinander verknüpft. Das Framework ist jedoch nur lokal auf den Endgeräten verfügbar. Die Lösung ist durch die Verwendung der ICMP-Nachrichten und die Erweiterung der Socketschnittstelle nicht auf beliebige Kontextwerte übertragbar.

2.4.3.3 Koordination ereignisbasierter Adaption

In [ECD+01, EFD+02a, EFD+02b] wird eine Plattform zur Koordination von Adaptionsoperationen verschiedener konkurrierender Anwendungen vorgestellt. Ausgangspunkt ist die Beobachtung, das Anwendungen, die Ressourcen gemeinsam nutzen (Endgerät, Netzwerkverbindung), ihre Anpassungsoperationen koordinieren müssen. Insbesondere unter Einbeziehen des Benutzers ermöglicht dies eine bestmögliche Ausnutzung der vorhande-

nen Ressourcen. Andernfalls könnten von den Anwendungen gegensätzliche oder ineffiziente Operationen ausgeführt werden (siehe z. B. [EFD+02a]).

Um eine Koordination der Adaption verschiedener autonomer Anwendungen zu erreichen, müssen Anwendungsentwickler dem Laufzeitsystem Informationen über die Adaptionsmechanismen und den Ausführungszustand ihrer Anwendung bereitstellen. Dazu werden Beschreibungen der Adaptionsmechanismen und deren Parameter sowie von Zustandsvariablen beim Laufzeitsystem registriert. Außerdem muss der Anwendungsentwickler Regeln für die Adaption in einer speziellen ereignisbasierten Sprache definieren [ECD+01]. Diese beschreiben Adaptionsmechanismen, die beim Auftreten bestimmter Events aufgerufen werden müssen. Dazu ist insbesondere ein systemweit eindeutiges Kontextsystem notwendig. Es erfolgt also eine Trennung der Adaptionsmechanismen und des Kontextes zur Aktivierung der Mechanismen. Das Laufzeitsystem überwacht den Zustand des gesamten Systems und trifft koordinierte Entscheidungen zur Adaption basierend auf den definierten Regeln. Ausgeführte Adaptionsoperationen werden dem Benutzer bekannt gegeben. Treten bei der Interpretation der Regeln Konflikte auf, wird der Benutzer in deren Auflösung einbezogen.

Die Lösung betrachtet mit der Koordination einen wesentlichen Aspekt der Adaption. Insbesondere wird eine Trennung der Adaptionsmechanismen von den Steuerinformationen vorgenommen. Die Lösung betrachtet jedoch Adaptionsmechanismen sehr grobgranular als Operationen, die durch ein Ereignis ausgelöst werden und den Zustand der Anwendung verändern. Insbesondere werden nur durch Ereignisse ausgelöste Adaptionsoperationen betrachtet. Diese adaptieren die Struktur der Anwendung, nicht jedoch Anwendungsdaten oder die Kommunikation.

2.4.4 Systemunterstützung für kontextabhängige Anwendungen

Die Nutzung von Kontext in Anwendungen stellt ohne Systemunterstützung einen erheblichen Aufwand dar. Je nach Art der Kontextwerte werden dabei Sensordaten ermittelt, Messungen durchgeführt, es wird auf Datenbanken zugegriffen oder der Benutzer befragt. Diese Rohdaten können zum größten Teil nicht direkt in Anwendungen eingesetzt werden, sondern müssen noch interpretiert, verknüpft und abstrahiert werden. Außerdem fallen die Rohdaten überwiegend auf verteilt an unterschiedlichen Orten und unterschiedlichen Geräten an und müssen entsprechend beschafft werden. Die Erzeugung und Vermittlung von Ereignissen stellt eine weitere Aufgabe bei der Kontextverarbeitung dar.

Da eine Vielzahl von Anwendungen von Kontextdaten profitieren kann, ist eine systembasierte Unterstützung dieser Standardaufgaben der Kontextverarbeitung wünschenswert. Anwendungsübergreifende Ansätze in Form eines Kontextframeworks oder Kontextdienstes ermöglichen eine Wiederverwendung von Code und Komponenten und verbergen außerdem die Details der Kontextverarbeitung. Vorhandene Lösungen unterscheiden sich vor allem in der Art des unterstützen Kontextes, der Generalität und dem Abstraktionsgrad (siehe auch [Lös02, Kad03]).

2.4.4.1 Kontextdienste für Ortsinformationen

Die Bedeutung von Ortskontext als meistgenutzte Kontextart spiegelt sich auch in der Verfügbarkeit von entsprechenden Kontextdiensten wider. Das Active Map System [ScT94] ist ein hierarchisch strukturiertes System für Orts- und Positionsinformationen. Damit können Informationen über den Aufenthaltsort von Geräten und Personen mit unterschiedlicher Abstraktion bzw. Genauigkeit ermittelt werden (z. B. in der Hierarchie: Regi-

on, Gebäude, Stockwerk, Raum). Die strenge Strukturierung ermöglicht insbesondere eine gute Skalierbarkeit des Systems. Dazu wurden weiterführende Betrachtungen durchgeführt (z. B. durch Gruppenbildung bei einer hohen Konzentration von Objekten an einem Ort).

Mobile Shadow [Fis03] basiert auf mobilen Softwareagenten und stellt eine verteilte Systeminfrastruktur für proaktive, ortssensitive Dienste zur Verfügung. Mobile Shadow arbeitet auf der Basis eines Abbildes der realen Welt, in das Nutzer und Zugangsrechner für WLAN abgebildet werden. Agenten bewegen sich in der virtuellen Welt mit dem Benutzer und agieren als Stellvertreter zur Beschaffung von Kontext und zum Auslösen von Ereignissen. Kontextinformationen können persistent gespeichert werden, bleiben aber auf den Ortskontext beschränkt.

2.4.4.2 Abstraktion der Stick-e Notes

Das Stick-e Note Framework [Pas97] verwendet die Metapher der Post-it Klebezettel. Virtuelle Klebezettel (Stick-e Notes) werden mit Objekten der realen Welt assoziiert und erlauben so die Zuordnung von Kontext zu diesen Objekten. Die Klebezettel enthalten Kontext und Inhalte. Der Kontext dient zur Formulierung von Bedingungen, zu denen der Klebezettel aktiviert wird. Der Inhalt legt fest, was bei einer Aktivierung geschehen soll (einfache Anzeige von Informationen, Auslösen einer Aktion). Eine Erweiterung des Konzeptes stellt der Context Information Service (CIS) [Pas98] dar. Über diesen können Anwendungen auf alle im CIS gespeicherten Informationen zugreifen. Im CIS wird die Welt als eine Menge von Entities modelliert, die einen Namen, Typ und weitere Informationen besitzen. Das Framework unterstützt mit diesem Ansatz vor allem die Verarbeitung von historischem Kontext und wurde unter anderem zur Realisierung einer Fieldwork-Anwendung zur Giraffenbeobachtung eingesetzt.

2.4.4.3 Allgemeine Abstraktionen für die Kontextverarbeitung und -aufbereitung

Das Context Toolkit [SDA99] stellt einen allgemeinen Dienst für kontextabhängige Anwendungen zur Verfügung. Kontext wird in Anlehnung an die Modellierung von Interaktionen zwischen graphischen Benutzeroberflächen und Anwendungen in Form von Kontext Widgets modelliert. Diese repräsentieren einen einzelnen Kontextwert und kapseln dessen Gewinnung bzw. Beschaffung. Die Implementierung eines Widgets bietet die Möglichkeit der Abfrage von Werten und der Historie sowie das Abonnement von Änderungsnachrichten. Neben dieser grundlegenden Abstraktion wurden weitere Abstraktionen zur Kontextverarbeitung definiert. Interpretatoren dienen der Verarbeitung und Verknüpfung einzelner Kontextwerte. Es können mehrere Kontextwerte als Eingabewerte definiert werden, die zu einem Ergebniswert verknüpft werden. Interpretatoren können von Widgets, Anwendungen, Aggregatoren und Interpretatoren aufgerufen werden. Aggregatoren repräsentieren den zu einem Entity gehörenden Kontext und agieren als Proxy für den Zugriff auf alle zu einem Entity gehörenden Widgets. Sie ermöglichen sowohl die Abfrage als auch das Abonnement von Informationen der von ihnen verwalteten Widgets.

Abstraktionen zur Gewinnung und Verknüpfung von Kontext werden auch im Projekt Technology for Enabled Awareness (TEA) [SAT+99] beschrieben. Ausgangspunkt ist die Sicht auf Kontext in einem drei-dimensionalen Modell, dessen Achsen die Umgebung (environment), der Benutzer (self) und die Aktivität (activity) sind. Auf dieser Basis wird eine Architektur zur Kontexterfassung vorgestellt, die aus den vier Schichten: Sensoren, Wahrnehmungselemente (Cue), Kontext und einer Anwendungsschicht besteht. Physikali-

sche Sensoren messen konkrete Werte in der Umgebung (Temperatur, Druck, Beschleunigung, Lichtintensität, Geräuschpegel, …). Logische Sensoren greifen auf gespeicherte Daten zu. Diese Rohdaten werden von den Wahrnehmungselementen aufbereitet. Jede Cue verarbeitet Werte genau eines Sensors. Die Cue erhält Wertefolgen genau eines Sensors zu diskreten Zeitpunkten und kann aus dieser Folge bestimmte Werte extrahieren oder die Werte miteinander verknüpfen. Funktionen von Cues sind z. B. Durchschnitt, Standardabweichung oder die Ermittlung der Basisfrequenz. Aus diesen aufbereiteten Daten kann auf die Situation eines Gerätes bzw. seines Benutzers geschlossen werden. Kontext beschreibt in diesem Ansatz sich gegenseitig ausschließende Situationen und je eine Wahrscheinlichkeit, mit der sich der Benutzer oder das Gerät in einer der Situationen befindet. Es wird eine Menge dieser Kontexte unterstützt, die Situationen zu verschiedenen Sachverhalten repräsentieren. Beispiele von Kontexten sind:

a) Benutzer hält Handy in der Hand oder dieses befindet sich auf einem Tisch oder in einer Tasche

b) ein Gerät wird in Bewegung oder in Ruhe genutzt

c) ein Gerät wird in einem Gebäude oder im Freien benutzt

d) ein Gerät wird in einem Auto, im Bus oder im Zug benutzt

e) das Gerät befindet sich in einem Auto das steht oder fährt.

Zur Entscheidung der aktuellen Situation werden logische Regeln definiert, die Cues miteinander verknüpfen. Anwendungen können die aktuelle Situation vom Kontextdienst abfragen und auf Basis von Skripten Aktionen auslösen. Dazu wurde eine Menge von Aktionen definiert, die aktiviert werden, wenn ein bestimmter Kontext betreten oder verlassen wird bzw. sich das Gerät oder der Benutzer in diesem Kontext befindet. Die Regeln zur Aktivierung werden vom Skriptprogrammierer in der Anwendung festgelegt. Außerdem können Skripts direkt auf Kontext zugreifen und so weiteres kontextabhängiges Verhalten implementieren. Die Erkennung von Situationen und die Adaption an diese wurden in zwei Anwendungen für PDA und GSM Handys getestet. Für PDAs wurde eine Notizanwendung um Kontext erweitert. Diese verändert in Abhängigkeit von der aktuellen Situation die Schriftgröße (z. B. große Schrift, wenn Benutzer in Bewegung ist oder es dunkel ist). Handys wurden um die Eigenschaft kontextabhängiger Profile erweitert. Je nach Bewegung des Benutzers, Aufenthaltsort und Situation wird ein entsprechendes Profil automatisch aktiviert (z. B. wenn Benutzer im Freien ist, dann Klingeltöne in höchster Lautstärke).

Sowohl im Context-Toolkit als auch in TEA werden mehrere Schritte auf unterschiedlichen Abstraktionsebenen zur Erfassung von Umgebungsinformationen und deren Verknüpfung und Interpretation zu Kontextwerten beschrieben. Während dieses Verarbeitungsmodell in TEA speziell für Messdaten von Sensoren eingesetzt wird, bietet das Context-Toolkit allgemeine Abstraktionen. Eine Verallgemeinerung nach [Lös02] wird in Abbildung 2-14 dargestellt. Kontextwerte werden demnach von Sensoren erfasst. Dies können Messdaten physikalischer Sensoren oder Werte logischer Sensoren sein (z. B. durch Extraktion, Benutzereingaben, Benutzerbeobachtung oder aus Datenbanken). Diese werden verknüpft und interpretiert, um allgemeine Kontextwerte zu ermitteln, die für Anwendungen verwertbar sind. Interpretation und Aggregation kann dabei kombiniert und in Folge angewendet werden. Anwendungen greifen auf diesen allgemeinen Kontext zu und führen auf dessen Basis weitere Abbildungen aus, um Aktionen auszulösen oder ihr Verhalten anzupassen.

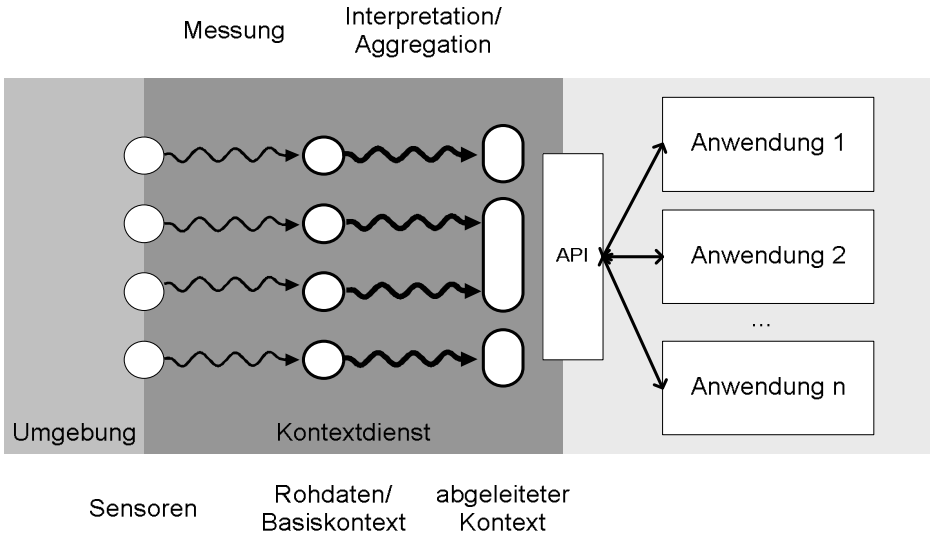


Abbildung 2-14: Schritte zur Gewinnung von Kontextinformationen (nach [Lös02])

2.4.5 Zusammenfassung

Wie in Kapitel 1 beschrieben, ist die Wahrnehmung der Ausführungsumgebung eine wesentliche Voraussetzung zur Adaption. Kontext bezeichnet Informationen, welche die Situation und Ausführungsumgebung von Anwendungen charakterisieren. Technische Informationen, die für die Adaption von Anwendungen in mobilen Infrastrukturen eine wesentliche Rolle spielen, können als Teilmenge des Gesamtkontextes einer Anwendungsumgebung betrachtet werden. Die Verfügbarkeit der Informationen über Endgeräte, Netzwerkverbindungen und dynamische Änderungen der Netzwerktopologie ist die Voraussetzung zur Adaption an die Anforderungen A1.1, A1.2 und A1.4 aus Kapitel 1, die technischer Natur sind. Die Anforderungen in Punkt A1.3 erfordern Informationen über den Benutzer und die Anwendungssituation. Diese werden traditionell in kontextabhängigen Systemen betrachtet und zur Verfügung gestellt. Kontext stellt somit die Quelle aller benötigten Informationen für die Adaption entsprechend der Anforderungen A1.1 bis A1.4 aus Kapitel 1 dar. Wie beschrieben, kann Kontext zum einen zur Steuerung und zum anderen zum Auslösen von Adaptionsoperationen verwendet werden. Eine Systemunterstützung in Form eines Kontextdienstes wurde in verschiedenen Projekten bereits untersucht und bietet eine Ausgangsbasis zur Realisierung adaptiver, kontextabhängiger Anwendungen. Insbesondere wurde bereits eine Reihe von Abstraktionen zur Kontextgewinnung untersucht. Diese dienen der schrittweisen Ermittlung von Kontext aus Rohdaten bzw. Basiskontext (siehe Abbildung 2-14). Diesen aufbereiteten Kontext können Anwendungen von einem Kontextdienst abfragen und führen dann weitere anwendungsabhängige Verarbeitungsschritte durch, um Kontext auf Aktionen oder ein kontextabhängiges Verhalten abzubilden.

In Projekten im Bereich Adaption wurde Kontext überwiegend nur implizit verwendet. Einige der Projekte betrachten jedoch eine explizite Nutzung von Kontext zur Adaption. Untersucht wurden dabei vor allem die Gewinnung von Informationen sowie Architekturen und Plattformen für deren Bereitstellung. Außerdem wurden vielfältige Adaptionsmechanismen betrachtet, durch die Anwendungen sich an eine bestimmte Systemumgebung anpassen können. Überwiegend wird aber eine Abbildung von Kontext auf konkrete Parameter von Adaptionsoperationen nicht betrachtet. Insbesondere existieren dazu nur Prototypen (z. B. [SAT+99]), die spezifische Lösungen für das kontextabhängige Verhalten von Anwendungen beschreiben. Der Zusammenhang zwischen Kontext und kontextabhängi-

gem bzw. adaptivem Verhalten stellt einen der Schwerpunkte der Arbeit dar. Ziel ist dabei die integrierte Betrachtung von Kontextabhängigkeit und Adaption sowie die Untersuchung von geeigneten Abstraktionen.

2.5 Entwurf und Beschreibung adaptiver Anwendungen und Systeme

Generell bestehen verteilte Anwendungen aus einer Menge von Komponenten, die durch die Herstellung von Kommunikationsbeziehungen zu einer Gesamtanwendung kombiniert werden. Sollen Anwendungen ihre Struktur, d. h. die eingesetzten Komponenten sowie deren Kommunikationsbeziehungen in Reaktion auf Veränderungen innerhalb ihrer Ausführungsumgebung anpassen können, spielt die flexible Beschreibung, Konstruktion und Änderung der Anwendungsstruktur eine wesentliche Rolle. Das Architekturmuster „Pipes and Filters“ wurde bereits im zweiten Teil des Kapitels als wesentliche Lösungsidee für eine flexible und lose Kopplung autonomer Adaptionsmechanismen identifiziert. Nachfolgend soll es deshalb kurz näher vorgestellt werden (Abschnitt 2.5.1). Softwarekomponenten zielen ebenfalls auf die Erzeugung autonomer Module, die miteinander kombiniert werden können. Sie werden in den meisten weiteren vorgestellten Konzepten als Grundelement betrachtet, die wesentlichen Ideen sollen deshalb in Abschnitt 2.5.2 vorgestellt werden. Der Ansatz der Konfigurationsprogrammierung stellt zur Beschreibung von Architekturen eine grundsätzliche Lösungsmöglichkeit dar. Architekturbeschreibungssprachen besitzen ähnliche Ziele, betrachten jedoch nur die Beschreibung von Architekturen als Ansatz einer grob-granularen Programmierung von Anwendungen. Die beiden genannten Ansätze werden in Abschnitt 2.5.3 und Abschnitt 2.5.4 diskutiert. Die Unified Modelling Language (UML) ist die gegenwärtige Standardsprache zur Softwareentwicklung. Insbesondere die Erweiterungsmöglichkeiten durch UML Profiles sowie die Definition von Meta-Modellen mittels der Meta-Object Facility Spezifikation (MOF) werden nachfolgend näher betrachtet (Abschnitt 2.5.5). Die Ansätze der Model Driven Architecture sowie der aspektorientierten Programmierung stellen neue Konzepte der Softwareentwicklung dar. Beide Ansätze bieten auch in Bezug zur Adaption neue Konzepte und werden deshalb in Abschnitt 2.5.6 und 2.5.7 kurz vorgestellt.

2.5.1 Das Entwurfsmuster „Pipes and Filters“

Das Entwurfsmuster „Pipes and Filters“ ist ein Architekturmuster zur Beschreibung der Struktur von Systemen, die Datenströme verarbeiten [BMR+96, VBT95]. Die Verarbeitung des Systems wird dabei in autonome Teilschritte zerlegt, die in Filterkomponenten gekapselt werden. Das Entwurfsmuster erlaubt eine Einkapselung von Teilfunktionalität in Filter sowie eine lose Kopplung der Filter über Pipes. Daraus resultieren eine größtmögliche Autonomie sowie eine einfache Rekombination der Filter. Das Entwurfsmuster eignet sich sehr gut zur Beschreibung der allgemeinen Struktur adaptiver Systeme. Es erlaubt die Modellierung aufeinander folgender Adaptionsschritte und unterstützt durch die lose gekoppelte Struktur eine flexible Kombinierbarkeit sowie die Wiederverwendbarkeit von Filtern.

In Abbildung 2-15 wird die grundlegende Struktur von Systemen entsprechend des Entwurfsmusters dargestellt. Eine Pipeline besteht nach [VBT95] aus einer Datenquelle, einer Datensenke sowie einer Reihe von Filtern. Zwischen den Filtern werden die Daten über sogenannte Pipes ausgetauscht. Filter lesen Daten über eine ankommende Pipe, verarbeiten diese Daten und senden die Verarbeitungsergebnisse über eine ausgehende Pipe. Jeder

Filter besitzt je eine Schnittstelle zum Empfangen von Daten (Reader) und zum Versenden von Daten (Writer). Die Schnittstellen sind getypt, Filter empfangen, verarbeiten und versenden also nur bestimmte Datentypen. Das Verhalten eines Filters in Bezug auf eine Schnittstelle kann entweder aktiv oder passiv sein. Eine Pipe koppelt die Writer-Schnittstelle eines Filters mit der Reader-Schnittstelle des nachfolgenden Filters. Eine der Schnittstellen muss dabei aktiv, die andere passiv sein. Entsprechend des Verhaltens der verbundenen Schnittstellen kann die Art der Anforderung der Daten in „push“ und „pull“ unterschieden werden. Werden ein aktiver Writer und ein passiver Reader miteinander verbunden, werden die Daten per „push“ vermittelt. Wird dagegen ein passiver Writer mit einem aktiven Reader verbunden, wird auf die Daten per „pull“ zugegriffen. Während der Datenfluss immer in der gleichen Richtung verläuft (von der Quelle zur Senke), kann der Steuerfluss dem Datenfluss entgegen verlaufen und seine Richtung von Pipe zu Pipe ändern. Es kann also in einer Pipeline zwischen Daten- und Steuerfluss unterschieden werden [VTB95].

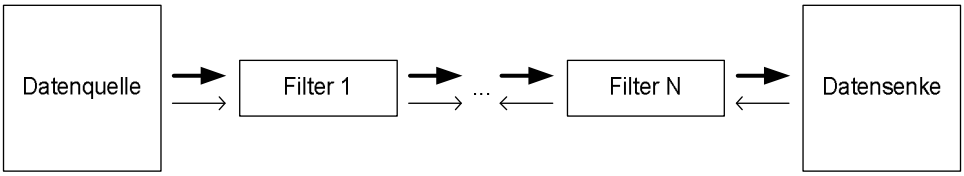


Abbildung 2-15: Struktur eines Systems entsprechend des Entwurfsmusters „Pipes and Filters“

2.5.2 Softwarekomponenten

Das Konzept von Komponenten als Bausteine für die Softwareentwicklung hat in den letzten Jahren eine weite Verbreitung gefunden. Diese Entwicklung wird nach [KoB98] vor allem durch die Notwendigkeit der Interoperabilität unabhängig voneinander entwickelter Softwaremodule und die kommerzielle Verfügbarkeit verteilter Plattformen wie CORBA und DCOM vorangetrieben. Die komponentenorientierte Softwareentwicklung verspricht eine einfache Entwicklung von Anwendungen durch das Zusammenfügen von (standardisierten) Bausteinen, die Wiederverwendbarkeit dieser Bausteine und eine leichte Änderbarkeit von Softwaresystemen durch die Weiterentwicklung bzw. den Austausch von Komponenten. Damit wird unter anderem der hohen Änderungsrate Rechnung getragen, der Softwaresysteme heute unterworfen sind.

Die Architektur, d. h. die Beziehungen zwischen Komponenten, rückt durch die komponentenorientierte Softwareentwicklung in das Zentrum der Betrachtungen [ChD00]. Diese Sichtweise ermöglicht die Veränderbarkeit und Anpassbarkeit von Software durch das Ersetzen von Komponenten entweder durch komplett neue Implementierungen oder neue Versionen aktueller Implementierungen. Individuelle Komponenten und deren Wiederverwendbarkeit sind ebenfalls von großer Bedeutung, stellen jedoch nach [ChD00] nicht die Hauptmotivation für Komponenten dar. Wesentliche Ziele der komponentenorientierten Softwareentwicklung sind also die Betrachtung der Architektur sowie die Ersetzbarkeit und Wiederverwendbarkeit von Komponenten.

Nicht zuletzt aufgrund der vielfältigen Anwendung des Begriffes Komponente und der damit verbundenen Technologien besteht keine Einigung auf eine Komponentendefinition. Die für eine Komponente wesentlichen Eigenschaften hängen insbesondere von der Betrachtungsweise ab. Nach [BrW98] können zwei prinzipielle Sichten auf Komponenten unterschieden werden. Die Entwurfssicht betrachtet abstrakte Komponenten als Grundlage für einen komponentenorientierten Softwareentwurf. Die Implementierungssicht betrachtet

Komponenten dagegen als fertige, kommerziell verfügbare Softwarebausteine, die in standardisierten Komponenten-Frameworks eingesetzt werden können. In [ChD00] werden diese beiden prinzipiellen Sichten verfeinert. Es werden die folgenden vier Sichten unterschieden:

1. Komponentenspezifikation: Diese Sicht definiert die „Hülle“ einer Komponente unabhängig von deren Implementierung. Spezifiziert werden die von der Komponente angebotenen und benötigten Schnittstellen und deren Beziehungen untereinander. Die Schnittstellen werden separat von Komponenten spezifiziert. Diese Sicht unterstützt vor allem eine Betrachtung der Architektur unabhängig von der Implementierung einzelner Komponenten.
2. Komponentenimplementierung: Diese Sicht definiert die Implementierung einer Komponente entsprechend einer gegebenen Spezifikation und damit den „Inhalt“ der durch die Spezifikation festgelegten „Hülle“. Damit wird eine klare Trennung der Spezifikation von der Implementierung einer Komponente erreicht. Eine Spezifikation kann verschiedene Implementierungen besitzen.
3. Installierte Komponente: Diese Sicht beschreibt die Installation von Komponentenimplementierungen in einer Komponentenplattform. Dieser Vorgang wird als Deployment bezeichnet und meldet die Komponentenimplementierung bei der Komponentenplattform an. Die Komponentenplattformen erlauben in dieser Sicht unter anderem das Hinzufügen von Informationen über die Nutzung von Plattformdiensten durch die Komponente (z. B. Transaktionsverarbeitung, Persistenz und Sicherheit).
4. Komponentenobjekt: Diese Sicht beschreibt eine Komponente zur Laufzeit. Komponentenobjekte repräsentieren Instanzen einer Komponente, die wie Objekte eine Identität besitzen und Zustandsinformationen und Funktionalität einkapseln.

Durch diese Sichten erfolgt eine klare Trennung der Spezifikation einer Komponente von deren Implementierung. Wesentlich für Komponenten ist nach [ChD00] außerdem der Bezug zu einem Komponentenstandard. Eine Komponente wird erst durch einen konkreten Bezug zu einer Komponente. Die folgende Definition einer Komponente von Szyperski soll für die weiteren Betrachtungen herangezogen werden:

> *„A component is a unit of composition with contractually specified interfaces and explicit context dependency only. A component can be deployed independently and is subject to composition by third parties.” [Szy02]*

Die Definition betont die ausschließlich expliziten Kontextabhängigkeiten sowie die unabhängige Verwendbarkeit von Komponenten. Sie besteht aus einem technischen und einem marktbezogenen Teil. Technisch gesehen ist eine Komponente eine Einheit zur Komposition, d. h. sie wird nur als ganzes spezifiziert, implementiert und eingesetzt. Sie besitzt vertraglich spezifizierte Schnittstellen. Verträge stellen für die komponentenorientierte Softwareentwicklung ein wichtiges Konzept dar. Sie werden zwischen mindestens zwei Parteien geschlossen und definieren sowohl die Rechte als auch die Verpflichtungen aller Parteien. In [ChD00] werden zwei Typen von Verträgen beschrieben: Nutzungsverträge und Realisierungsverträge.

Nutzungsverträge beziehen sich auf Schnittstellenspezifikationen. Die beteiligten Parteien sind der Anbieter und der Nutzer einer Schnittstelle. Die Spezifikation der Schnittstelle stellt einen Vertragsentwurf dar, der Vor- und Nachbedingungen, Invarianten sowie weitere nicht-funktionale Anforderungen definiert. Der Vertrag sieht vor, dass der Nutzer der

Schnittstelle sicherstellen muss, dass die Vorbedingungen erfüllt sind. Ist dies der Fall, müssen vom Anbieter die Nachbedingungen erfüllt werden ohne die Invarianten und geltende nicht-funktionale Anforderungen zu verletzen. Realisierungsverträge werden zwischen einer Komponentenspezifikation und möglichen Implementierungen geschlossen. Die Komponentenspezifikation definiert die angebotenen und die benutzten Schnittstellen einer Komponente sowie deren Zusammenhänge. Diese Spezifikation muss von jeder Implementierung erfüllt werden.

Explizite Kontextabhängigkeiten beschreiben neben den von der Komponente benutzten Schnittstellen auch Beziehungen zu einem Komponentenmodell und Komponentenplattformen. Das Komponentenmodell definiert die von der Komponente geforderten Schnittstellen sowie Regeln zur Komposition von Komponenten. Eine Komponentenplattform stellt die Laufzeit- und Dienstumgebung für Komponenten eines Komponentenmodells zur Verfügung. Damit verbunden definiert die Komponentenplattform Regeln für das Deployment, die Installation, die Aktivierung und Abarbeitung von Komponenten. Explizite Kontextabhängigkeiten sind damit neben den benutzten Schnittstellen auch das zugrunde liegende Komponentenmodell sowie Informationen über die Komponentenplattform und die Implementierung der Komponente (z. B. Implementierungssprache, Deploymentinformationen, verwendeter Compiler, usw.).

Im Folgenden werden einige Komponentenmodelle vorgestellt. Diese spezifizieren Komponenten aus der Implementierungssicht und setzen überwiegend auf verteilten Objektsystemen auf.

2.5.2.1 Enterprise JavaBeans

Enterprise JavaBeans (EJB) ist ein serverseitiges Komponentenmodell auf der Basis der Programmiersprache Java. Der Laufzeitumgebung liegen die Konzepte eines komponentenorientierten Transaktionsmonitors sowie der verteilten Objekttechnologie zugrunde. Aktuell ist die Version 2.1 der EJB Spezifikation [Sun03]. Die Spezifikation definiert drei fundamentale Komponententypen: Session Beans, Entity Beans und Message Driven Beans. Session und Entity Beans kommunizieren basierend auf RMI, während Message Driven Beans asynchrone Nachrichten entsprechend der Java Message Service (JMS) Spezifikation verarbeiten.

1. Session Beans stellen zum einen für Clients bzw. weiteren Vermittlungsschichten (z. B. Servlets oder JavaServer Pages (JSP)) einen Interaktionspunkt mit der Anwendungslogik dar. Zum anderen enthalten Session Beans Teile der Anwendungslogik und implementieren insbesondere die Logik zur Verwaltung und Kombination von Entity Beans. Es können zwei Basistypen von Session Beans unterschieden werden. Stateless Session Beans beinhalten eine Menge logisch zusammengehörender Methoden. Jede Methode hängt nur von ihren Eingabeparametern ab und arbeitet unabhängig vom Zustand der Sitzung. Statefull Session Beans enthalten dagegen Informationen über die aktuelle Sitzung. Das Ergebnis von Methodenaufrufen ist neben den Eingabeparametern noch vom aktuellen Zustand der Bean abhängig (conversational state).
2. Entity Beans repräsentieren Anwendungsdaten, die dauerhaft gespeichert werden (z. B. Daten einer relationalen Datenbank). Eine Entity Bean stellt demnach z. B. eine Tabelle in einer Datenbank dar, die Instanzen entsprechen den Zeilen dieser Tabelle. Während Session Beans separate Clientsitzungen verwalten und deren aktuellen Status sichern, können verschiedene Clients über unterschiedliche Session Beans auf ein und dieselbe Entity Bean Instanz bzw. auf die von dieser repräsentierten Daten zugreifen. Dabei übernimmt die Session Bean die Transaktionskontrolle.

3. Message Driven Beans sind zustandslose Komponenten zur Verarbeitung von Ereignissen des Java Messaging Service. Die Beans implementieren keine Methoden, die von anderen Beans aufgerufen werden können, sondern abonnieren beim JMS bestimmte Ereignisse. Ihre Schnittstelle besteht aus der Methode onMessage(), die aufgerufen wird, wenn Ereignisse für die Bean vorliegen. Message Driven Beans können während ihrer Verarbeitung andere Session bzw. Entity Beans aufrufen.

Die Laufzeitumgebung für Enterprise Java Beans wird durch einen Container bereitgestellt. Dieser verwaltet ein oder mehrere Beans und deren Instanzen, steuert deren Abarbeitung, d. h. er verwaltet deren Lebenszyklus, und ermöglicht den Zugriff auf Systemdienste über spezifizierte Schnittstellen. Die vom EJB-Container bereitgestellten Systemdienste sind Transaktionsmanagement, Sicherheit, entfernte Kommunikation zwischen Komponenten, ein Namensdienst sowie Persistenz für Entity Beans. Die Dienste zur Transaktionssteuerung und Persistenz können komplett durch den Container verwaltet oder vom Komponentenentwickler selbst implementiert werden (container/bean managed persistence/transactions)

Die Erstellung einzelner Beans erfolgt durch einen Bean Provider. Dieser stellt die Klassen und weitere Daten zu einem Archiv zusammen und definiert einen Deployment Descriptor, der die Bestandteile einer Bean sowie Informationen zur Persistenz von Entity Beans enthält. Die Komposition von Beans zu größeren Anwendungseinheiten wird vom Application Assembler durchgeführt. Dieser erstellt dazu einen so genannten Assembly Descriptor. Dieser ist Teil des Deployment Descriptors und enthält Transaktionsattribute und Sicherheitsdefinitionen. Durch diese Attribute beschreibt der Assembly Descriptor Beziehungen zwischen Beans. Die Beziehungen selbst werden im Quellcode der Beans implementiert.

Das EJB Komponentenmodell definiert einen Vertrag zwischen Beans und dem Container. Dieser ist in Form der Verantwortlichkeiten des Containers gegenüber der Bean spezifiziert. Entsprechend des Vertrages und ihres Typs muss eine Bean eine Reihe von Methoden enthalten, die vom Container an definierten Punkten des Lebenszyklus' der Bean aufgerufen werden. Außerdem besitzen Session und Entity Beans zwei Typen von Schnittstellen, über die Instanzen verwaltet (Home-Schnittstelle) und auf die implementierte Anwendungslogik zugegriffen werden kann (Remote Schnittstelle). Ab Version 2.0 der EJB-Spezifikation [Sun01] können beide Schnittstellen sowohl lokal als auch entfernt aufgerufen werden. Damit existieren die vier Schnittstellen Remote, Remote home, Local und Local home.

2.5.2.2 CORBA Component Model

Das CORBA Component Model (CCM) ist ein serverseitiges Komponentenmodell [OMG02A]. Es ist seit der Version 3 Teil der CORBA Spezifikation der Object Management Group (OMG) [OMG02b]. CORBA, einschließlich des Komponentenmodells, ist sowohl von Programmiersprachen als auch Plattformen unabhängig. Das Komponentenmodell wurde in Anlehnung an EJB entwickelt und stellt eine logische Erweiterung dieses Komponentenmodells dar. Insbesondere soll eine vollständige Kompatibilität zu bestehenden EJB-Lösungen unterstützt werden. Derzeit existieren jedoch noch keine Implementierungen, obwohl der Standard bereits seit 2001 in einer nahezu vollständigen Version zur Verfügung steht.

Die Verwandtschaft zu EJB wird an den von CCM unterstützten Komponententypen deutlich. Dies sind die vier Typen Service, Session, Entity und Process. Service Komponenten

repräsentieren zustandslose Sessions und entsprechen damit Stateless Session Beans. Der Entity Typ korrespondiert mit Entity Beans. Sessionkomponenten können über eine Transaktion Zustand enthalten und entsprechen damit Stateful Session Beans. Eine Erweiterung gegenüber EJB stellen Process Komponenten dar, die persistente Sitzungen repräsentieren. Message-Driven Beans besitzen dagegen keine direkte Entsprechung, in CCM können alle Komponententypen Schnittstellen sowohl zum Empfangen als auch zum Versenden von Ereignissen besitzen.

Das Komponentenmodell sieht für CORBA Components einer Reihe von Schnittstellen vor (siehe Abbildung 2-16). Die Funktionalität einer Komponente wird durch Ports spezifiziert, die von einem der Typen Facet, Receptacle, Event Source und Event Sink sein können. Facets repräsentieren von der Komponente angebotene, Receptacles die benötigten Schnittstellen. Eine Facet-Schnittstelle ist also mit der Remote-Schnittstelle einer Bean vergleichbar, Receptacle-Schnittstellen werden von EJB jedoch nicht unterstützt. Event Sources und Event Sinks beschreiben die benötigten und angebotenen ereignisbasierten Schnittstellen. Während Facets und Receptacles direkt miteinander verbunden werden, erfolgt die ereignisbasierte Kommunikation immer indirekt über CORBA Event Channels. Außerdem können Komponenten Attribute enthalten, die über entsprechende Operationen gelesen und verändert werden können. Attribute sind vorrangig zur Konfiguration der Komponenten vorgesehen, bleiben aber nicht auf diese Funktion beschränkt. Anders als bei EJB können CORBA Components mehrere Schnittstellen implementieren. Jede Facet wird durch eine unterschiedliche Objektreferenz repräsentiert, über die Clients auf die jeweilige Schnittstelle zugreifen können. Zur Verwaltung der Facets stellt jede Komponente eine übergeordnete Schnittstelle (Equivalent Interface) zur Verfügung. Diese erlaubt den Zugriff auf Informationen über die Eigenschaften der Komponente (Introspection) und die Navigation zwischen Facets. Das Komponentenmodell unterscheidet zwei Arten von Komponenten: Basiskomponenten und erweiterte Komponenten. Basiskomponenten enthalten keine Ports, sondern nur Attribute. Erweiterte Komponenten können dagegen alle Porttypen beinhalten.

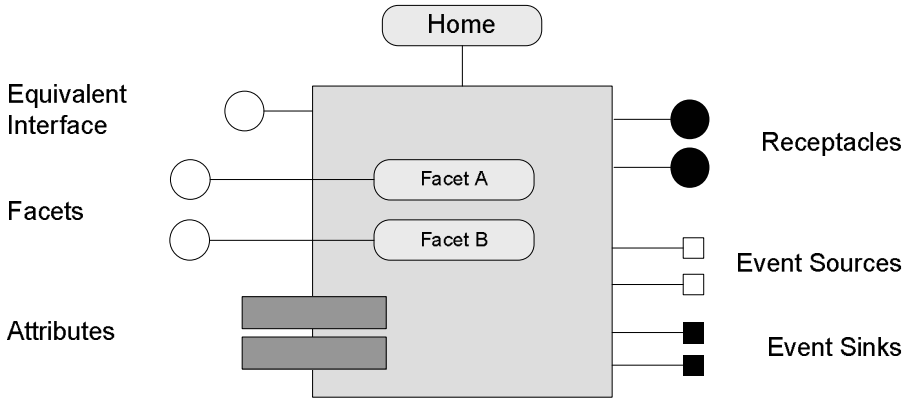


Abbildung 2-16: Bestandteile einer CORBA Komponente

Die Beschreibung von Komponenten und deren Komposition erfolgt durch mehrere Descriptoren. Software Package Descriptoren beschreiben Softwarepakete, die eine Komponente und eine oder mehrere alternative Implementierungen dieser Komponente enthalten. Ein Softwarepaket kann sowohl als Archiv als auch in separaten Dateien vorliegen. Der Descriptor enthält eine Beschreibung der Komponente und deren Implementierungen, deren Speicherorte, die Schnittstelle in IDL sowie Details zu den einzelnen Implementierungen wie Betriebssystem, Programmiersprache und Compiler. Die Implementierungen enthalten jeweils einen CORBA Component Descriptor. Dieser beschreibt die Spezifikation sowie

Informationen zum Deployment einer Komponente. Enthalten sind die Attribute und Ports der Komponente, Informationen zu Vererbungsbeziehungen zwischen Komponenten und dem Repository der Komponente. Deployment-Informationen umfassen den Typ der Komponente und die Art und Weise der verwendeten Containerdienste (z. B. Transaktion, Persistenz und Threading).

Die Komposition wird durch einen Assembly Descriptor auf Instanzenebene beschrieben. Enthalten sind Beschreibungen der einzelnen Komponenten sowie Verbindungs- und Verteilungsinformationen. Verbindungen können im Assembly Descriptor deklarativ durch die Verknüpfung entsprechender provides und requires-Schnittstellen bzw. Event Sources und Sinks hergestellt werden. Verbindungen können auch zur Laufzeit durch Aufrufe innerhalb der Komponenten erzeugt werden. Diese werden durch den Assembly Descriptor nicht erfasst. Außerdem kann die Platzierung der Komponenteninstanzen definiert werden.

Die Laufzeitumgebung für CORBA Components wird auf Basis von CORBA ebenfalls durch einen Container bereitgestellt. Damit stehen den Komponenten alle CORBA Services und CORBA Facilities zur Verfügung. Die Verwaltung des Lebenszyklus' einer Komponente erfolgt über deren Home-Schnittstelle.

2.5.2.3 COM+

Mit Windows 95 präsentierte Microsoft das Component Object Model (COM) zur Erstellung von Softwarebausteinen, die lokal innerhalb einer Windows-Plattform miteinander kommunizieren konnten. Mit DCOM fand eine Erweiterung der Plattform um die Möglichkeit der entfernten Kommunikation zwischen Komponenten statt. Parallel dazu wurde der Microsoft Transaction Server (MTS) und der Microsoft Message Queue (MSMQ) entwickelt. Durch die Integration von DCOM, MTS und MSMQ zu COM+ wurde die Komponentenplattform um wesentliche Dienste erweitert (siehe [EdE99]).

Das Komponentenmodell von COM+ definiert nur einen allgemeinen Komponententyp. Die Anforderungen an Komponenten werden auf Binärebene beschrieben. COM+ Komponenten sind somit unabhängig von Programmiersprachen und Plattformen. In der Praxis existiert eine weitreichende Unterstützung für COM+ jedoch nur auf der Windows-Plattform. Komponenten können in COM+ ebenso wie CORBA Components mehrere Schnittstellen anbieten. Die benutzten Schnittstellen werden dagegen nicht spezifiziert. Jede Komponente implementiert die IUnknown-Schnittstelle, die wie das Equivalent Interface des CCM eine Navigation zu den weiteren Schnittstellen ermöglicht. Außerdem besitzt jede Schnittstelle eine Methode QueryInterface, über die auf alle Schnittstellen einer Komponente über einen logischen Namen zugegriffen werden kann. Eine Erweiterung der IUnknown Schnittstelle stellt die IDispatch Schnittstelle dar. Diese enthält weitere Methoden, die eine dynamische Aufrufgenerierung zur Laufzeit ermöglichen (ähnlich CORBA).

Die Laufzeitumgebung von COM+ stellt auf der Grundlage des MTS sowie weiterer integrierter Dienste eine Unterstützung für Transaktionen, Sicherheit, Administration, Queue-basierte asynchrone Nachrichten, Ressource Pooling und Just-in-Time Aktivierung bereit. Persistenz wird in COM+ nicht direkt unterstützt. Es werden nur die Zustandsdaten ohne Bezug zur Identität der COM+ Objekte gespeichert. Auf Basis dieser Daten können beliebig viele Kopien einer gespeicherten Komponente mit verschiedenen Identitäten erzeugt werden.

2.5.2.4 Vergleich von Komponentenplattformen

Tabelle 2-4 stellt die Komponentenmodelle EJB, CCM und COM+ vergleichend gegenüber. Wesentlich für die betrachteten Modelle ist der starke Bezug zur Objektorientierung, zum Teil haben sich die Modelle auf Basis dieser Technologie entwickelt. Die Komponentenmodelle und die entsprechenden Plattformen unterstützen vor allem die Implementierung von Komponenten und stellen dazu wichtige Dienste für verteilte Umgebungen wie Transaktionsmanagement, Sicherheit, Persistenz und Naming bereit. Die Komponenten besitzen Objekteigenschaften, insbesondere erfolgt die Verbindung von Komponenten durch die Ermittlung von Referenzen im Code. Schnittstellen werden durch entsprechende Beschreibungssprachen (z. B. CORBA IDL) definiert. Die Definition der benutzten Schnittstellen und die Deklaration von Verbindungen werden nur von CCM unterstützt. Die Spezifikation von Komponenten und Verbindung zwischen diesen auf Basis expliziter Kontrakte ist aber in keiner der Komponentenplattformen möglich.

	EJB	CORBA Components	COM+
Einsatz	serverseitige Komponenten, verteilte Anwendungen, plattformunabhängig	serverseitige Komponenten, verteilte Anwendungen, Plattformunabhängig	plattformunabhängig, jedoch hauptsächlich Microsoft Windows
Programmiersprachen	Java	unabhängig, mehrere Language-Bindungs, CORBA IDL	unabhängig, Komponentenmodell mit Anforderungen an übersetzte Komponenten (Binaries)
Komponententypen	Session, Entity, Message-Driven	Service (stateless session beans), Session (stateful session beans), Entity (entity beans), Process (persistente Session)	ein allgemeiner Komponententyp
Kommunikationstypen	RMI, Events (JMS)	RPC, Events	RPC (DCOM), Events, MessageQueue
Schnittstellenspezifikation	als Java interface, nur provides-Schnittstellen, nur eine Remote-Schnittstelle pro Bean	CORBA IDL, provides (Facets) und requires-Schnittstellen (Receotacles), mehrere Schnittstellen pro Komponente	COM IDL, nur provides-Schnittstellen, mehrere Schnittstellen pro Komponente
Komponentenmodell	call-back Methoden für Lifecycle Management, lokale und entfernte Home und Remote-Schnittstellen	Home- und Equivalent-Schnittstellen, Ports (Facets, Receptacles, Event Source, Event Sink), Attributes	IUnknown Schnittstelle zur Navigation zwischen Schnittstellen, Methode QueryInterface als Methode aller weiteren Schnittstellen
Komposition	Deployment-Descriptor (Persistenz, Sicherheit, Transaktionen, Naming), enthält Assembly-Descriptor	Software Package Descriptor, CORBA Component Descriptor, Assembly Descriptor, Property File Descriptor	kein Konzept
Containerdienste	Naming, Persistenz, Transaktion, Sicherheit, entfernte Komm., Lebenszyklus	CORBA Services (Naming, Persistenz, Events, Transaktionen, Sicherheit, Aktivierung, Lebenszyklus, …), CORBA Facilities	Transaktionen, Sicherheit, Events, Queue-basierte asynchrone Nachrichten, Ressource Pooling, Just-in-Time Aktivierung

Tabelle 2-4: Vergleich der Komponentenplattformen EJB, CCM und COM+

2.5.3 Konfigurationsprogrammierung

Der Ansatz der Konfigurationsprogrammierung (configuration programming) hat die Unterstützung der Konstruktion konfigurierbarer und erweiterbarer paralleler und verteilter

Systeme zum Ziel. Diese können als eine Menge von Komponenteninstanzen betrachtet werden, zwischen denen Kommunikationsbeziehungen bestehen. Viele Komponentenmodelle sind eng mit einer oder mehreren Programmiersprachen gekoppelt. Damit werden die Beziehungen zwischen den Komponenten, d. h. die Struktur der Anwendungen implizit im Programmcode festgelegt. Aus diesem Grund sind Strukturänderungen nur durch Änderungen der Programmkomponenten möglich und somit sehr aufwendig.

In Conic [MKS89, KMN89] sowie den Nachfolgern Rex [MKS+90, KMS+92], Regis [MDK94, MDE+95] und Koala [OLK+00] erfolgt die Beschreibung der Konfiguration von Anwendungssystemen explizit in einer dedizierten Sprache (configuration language), getrennt von der Art der Kommunikationsbeziehungen zwischen den Komponenten und der Implementierungssprache der einzelnen Komponenten. Es erfolgt also eine Trennung der Beschreibung von Einzelkomponenten, Kommunikation und Struktur der Anwendungssoftware. Zum einen kann damit der Komponentenentwickler keine Annahmen über die Konfiguration treffen, in der eine Komponente später eingesetzt wird, zum anderen ist es dem Anwendungsentwickler nicht erlaubt, die Interna einer Komponente zu ändern, um sie an eine bestimmte Konfiguration anzupassen. Mit diesem Ansatz werden Strukturänderungen ohne Änderung der einzelnen Komponenten insbesondere auch zur Laufzeit möglich. Die Systeme besitzen die folgende Grundstruktur:

- Programmiersprache für Programmmodule (module programming language) in Conic bzw. eine von der Programmiersprache unabhängige Schnittstellenbeschreibungssprache (IDL)
- Konfigurationssprache (Conic bzw. Darwin in Rex und Regis)
- Nachrichtenprimitive zur Kommunikation zwischen Modulen in Conic und Rex bzw. benutzerdefinierte Kommunikationsdienste (services) in Regis
- Konfigurationsmanager und Laufzeitsystem

Komponenten

Komponenten kommunizieren über wohldefinierte Schnittstellen mit ihrer Umgebung. Die IDL ermöglicht die Definition von Schnittstellen unabhängig von der Implementierungssprache der Komponenten. Schnittstellen werden anhand ihrer Funktionen (Ports) sowie der Typen der über diese Funktionen ausgetauschten Daten beschrieben. Anhand der Schnittstellendefinitionen können Komponenten definiert werden. Für jede Komponente werden sowohl die Schnittstellen, die sie anderen Komponenten zur Verfügung stellen (provides), als auch die von der Komponente benutzten Schnittstellen (requires) beschrieben. Insbesondere besitzt eine Komponente keine direkten Beziehungen mehr zu anderen Komponenten. Der Zugriff auf andere Komponenten erfolgt mit Hilfe lokaler Namen indirekt über die als „requires“ definierten Schnittstellen. Damit wird der Einsatz der Komponenten unabhängig von dem Kontext möglich, in dem sie später ausgeführt werden sollen (siehe auch [MKS+90, OLK+00]).

Kommunikation

Zur Festlegung der Kommunikationsbeziehungen zwischen Komponenten wurden in Conic und Rex eine Menge von Nachrichtenprimitiven bereitgestellt, mit denen in der Komponentenimplementierung die Art des Datenaustausches für die einzelnen Funktionen (Ports) festgelegt werden konnte [MKS89, MKS+90]. Die Nachrichtenprimitive ermög-

lichten im Wesentlichen die Definition asynchroner und synchroner Aufrufe. Synchrone Aufrufe können beim Client blockierend oder parallel mit Hilfe von Promises [Lis88] realisiert werden. Außerdem existieren Mechanismen zur Definition von Zeitbegrenzungen für Aufrufe über Timeouts und zur Fehlerbehandlung.

Die Festlegung der Primitive beschränkt die Komponenten jedoch auf eine begrenzte Anzahl von Kommunikationsmechanismen. Außerdem wurden diese auf der Ebene der Programmiersprache definiert. In Regis wird diese Beschränkung durch die Trennung von Komponentenrealisierung und Kommunikation aufgehoben. Die Kommunikationsbeziehungen werden in diesem System durch Kommunikationsobjekte hergestellt, die die Interna des Nachrichtenaustausches kapseln [MDK94]. Den vom System bereitgestellten Kommunikationsobjekten kann der Benutzer selbst definierte Objekte hinzufügen, mit denen beliebige Kommunikationsmechanismen realisierbar sind. Die Kommunikationsobjekte werden in der Komponentendefinition den Schnittstellen der Komponente zugewiesen. Diese werden somit durch den Typ der Kommunikationsbeziehung (z. B. Port oder Strom) sowie die Typen der ausgetauschten Daten beschrieben.

Konfiguration

Verteilte Programme bestehen aus einer Menge von Komponenten, die durch die Herstellung von Kommunikationsbeziehungen zu einer Anwendung komponiert werden. Kommunikationsbeziehungen werden durch die Bindung von Komponentenschnittstellen erzeugt. Requires-Schnittstellen müssen dabei an genau eine provides-Schnittstelle gebunden werden, provides-Schnittstellen können dagegen an null oder mehr requires-Schnittstellen gebunden werden. Vor der Verbindung der Schnittstellen findet eine Überprüfung des Schnittstellentyps sowie der Typen der ausgetauschten Daten statt. Diese müssen übereinstimmen, damit eine Bindung hergestellt werden kann.

Zusammengesetzte Komponenten können aus Instanzen von Basiskomponenten (in Koala werden diese als Module realisiert [OLK+00]) sowie anderen zusammengesetzten Komponenten erzeugt werden. Dadurch entstehen hierarchische Komponentenkompositionen, auf deren höchster Ebene die Komponente steht, die die gesamte Anwendung beschreibt. Mittels der Konfigurationssprache werden zusammengesetzte Komponenten durch die enthaltenen Komponententypen sowie die aus diesen erzeugten Instanzen und die Schnittstellenbindungen zwischen den Instanzen beschrieben. Zur Erzeugung komplexer Konfigurationen stehen spezielle Schlüsselworte zur Verfügung, die die Erzeugung von Feldern von Instanzen gleichen Typs sowie die Bindung deren Schnittstellen durch Anweisungswiederholungen (z. B. innerhalb einer for-Schleife) ermöglichen. Die Platzierung der Instanzen wird anhand logischer Orte beschrieben, die durch die Ausführungsumgebung auf physikalische Orte (Prozessoren) abgebildet werden [MDK94]. Dabei können auch Zusammenhänge zwischen der Platzierung mehrerer Komponenten definiert werden. Beispielsweise kann der Aufenthaltsort einer Komponente an den einer anderen Komponente gebunden werden, wodurch beide Komponenten am gleichen Ort platziert werden.

Die Konfigurationsbeschreibung wird durch das Laufzeitsystem interpretiert. Dieses erzeugt ausgehend von der Komponente auf höchster Ebene rekursiv die Instanzen aller beschriebenen Komponenten und platziert diese entsprechend auf den Rechnern im Verwaltungsbereich des Laufzeitsystems. Im Unterschied zu seinen Vorgängern, Conic [MKS89] und Rex [KMS+92], in denen Konfigurationsbeschreibungen zentralisiert und sequentiell interpretiert wurden, besitzt Regis ein verteiltes Laufzeitsystem, das eine nebenläufige Auswertung der Systemkonfiguration ermöglicht [MDE+95]. Die physische

Verteilung konnte damit vollständig orthogonal von der logischen Struktur spezifiziert werden.

Dynamische Konfiguration

Conic unterstützte nur die Definition statischer Konfigurationen. Mit Hilfe des Konfigurationsmanagers konnten über das Laufzeitsystem jedoch interaktiv Änderungen initiiert werden [KMN89]. In Rex können zusätzlich zur interaktiven Änderung des Systems Strukturänderungen in der Konfigurationssprache definiert werden [MKS+90]. Dazu stehen inverse Operationen zur Entfernung von Instanzen und Bindungen aus dem System zur Verfügung. Die Änderungen können von den Komponenten zur Laufzeit aufgerufen werden. Damit beschreibt die Konfigurationsdefinition jedoch nur noch die Struktur des Systems zur Initialisierungszeit. Zur Laufzeit sind beliebige Änderungen der Konfiguration möglich.

In Regis werden deshalb die Möglichkeiten der Strukturänderungen eingeschränkt [MDK94]. Es stehen nur noch die beiden Methoden Lazy Instantiation (in Verbindung mit einer rekursiven Strukturdefinition) sowie die dynamische Instantiierung zur Verfügung. Damit wird die potentielle Struktur des Systems beschrieben, die zur Laufzeit jedoch nicht vollständig aufgebaut werden muss. Diese Methoden verlagern jedoch einen Teil der Strukturinformationen aus der Konfigurationsbeschreibung in den Programmcode. Die Reduzierung der Modifikationsmöglichkeiten erhöht jedoch die Klarheit der Strukturbeschreibung. Zur Wiederverwendung von Komponentenstrukturen können in Regis außerdem generische Komponenten definiert werden, d. h. Komponentenstrukturen, deren Typen parametrisierbar sind. Beispielsweise kann auf diese Weise eine Baumstruktur definiert werden, deren Knoten bestimmten Schnittstellen, nicht jedoch einem bestimmten Typ genügen müssen. Der Typ wird in der Definition durch eine Typvariable beschrieben, die erst durch den Compiler anhand der definierten Bindungen an einen bestimmten Typ gebunden wird. Generische Komponenten beschreiben nicht nur die logische Struktur einer Komponente sondern auch, wie diese Struktur auf ein reales System abgebildet werden soll.

Koala besitzt außerdem Erweiterungen zur Unterstützung der Flexibilität in Komponentendefinitionen. Beispielsweise können Schnittstellen auch gebunden werden, wenn diese nicht gleichen Typs sind. Diese sind kompatibel, wenn die provides-Schnittstelle alle Funktionen der requires-Schnittstelle enthält (interface compatibility). In ähnlicher Weise können Funktionen mit unterschiedlichen Namen aneinander gebunden werden, wenn diese typkompatibel sind. Weitere Möglichkeiten der Überwindung von Inkompatibilitäten zwischen Komponenten sind in [OLK+00] beschrieben.

2.5.4 Architekturbeschreibungssprachen

Architekturbeschreibungssprachen (ADLs) ermöglichen den Entwurf von Softwaresystemen auf einer höheren Abstraktionsebene als der von Modulen oder Code in Programmiersprachen. Im Allgemeinen werden Softwarearchitekturen anhand der Komponenten (dem Ort der Verarbeitung), den Kommunikationsverbindungen zwischen Komponenten (der Kommunikation) und der Konfiguration, d. h. der Verknüpfung von Komponenten über Kommunikationsverbindungen, beschrieben. Damit sind Architekturbeschreibungen Modelle bzw. Abstraktionen von der Implementierung des zu entwerfenden Systems. Um eine Architekturbeschreibung in ein reales System zu überführen, ist deshalb eine Abbildung der Architektur auf eine Implementierung notwendig. Diese kann durch eine Werkzeugun-

terstützung zum großen Teil automatisiert werden. Insbesondere sind damit verschiedene Abbildungen entsprechend der Eigenschaften unterschiedlicher Plattformen und Laufzeitsysteme möglich.

Die Beschreibung der Architektur erfolgt aus einer bestimmten Sicht auf Softwaresysteme. Die Abstraktion von Plattform- und Implementierungsdetails ermöglicht eine Fokussierung auf die Struktur des Softwaresystems. Die Sicht ermöglicht durch die Betrachtung von Komponenten als unabhängige Verarbeitungselemente und der expliziten Beschreibung der Aspekte der Kommunikation durch Konnektoren eine Entkopplung von Verarbeitung und Kommunikation im System. Damit wird ein hoher Grad der Wiederverwendung und Rekonfigurierbarkeit erreicht. Die abstrakte Sicht auf die Architektur erlaubt ein „Programmieren im Großen“ und unterstützt damit insbesondere den Entwurf komplexer Systeme sowie die Entwicklung von Architekturfamilien. Dies führt nicht zuletzt zu einer Kostenreduzierung beim Softwareentwurf [HNS00, MeT00].

Gegenwärtig existiert eine große Zahl von Architekturbeschreibungssprachen mit zum Teil sehr unterschiedlichen Zielen und damit unterschiedlichen Sprachansätzen. Mit ACME [GMW97] wurde eine Sprache entwickelt, die einen Austausch von Architekturbeschreibungen zwischen unterschiedlichen Sprachansätzen und Werkzeugen ermöglichen soll. Diese besteht zum einen aus einer Grundmenge von Elementen zur Beschreibung der Struktur von Softwaresystemen, zum anderen enthält sie ein Konzept zum Hinzufügen zusätzlicher architekturspezifischer Informationen. ACME ist somit sowohl eine Schnittmenge (Strukturbeschreibung) als auch eine Vereinigung (Erweiterungen) von ADLs. Zur Definition von Strukturen enthält ACME sieben Basiselemente: Komponenten, Ports, Konnektoren, Rollen, Konfigurationen, Repräsentationen und Repräsentation Maps (siehe Abbildung 2-17).

Komponenten repräsentieren die primären Elemente der Verarbeitung und Datenspeicherung einer Architektur. Konnektoren beschreiben die Interaktionen zwischen den Komponenten. Sie verbinden damit unabhängige Komponenten zu einer Architektur. Komponenten besitzen Schnittstellen, die als Ports bezeichnet werden. Jeder Port definiert dabei einen bestimmten Punkt der Interaktion. Ports umfassen sowohl die von der Komponente implementierten als auch die von der Komponente aufgerufenen Schnittstellen.

Konnektoren repräsentieren die primären Elemente der Kommunikation und Interaktion einer Architektur. Sie besitzen ebenfalls Schnittstellen, die als Rollen definiert werden. Eine Rolle repräsentiert dabei einen Teilnehmer der zugehörigen Interaktion (z. B. „Client“ und „Server“ oder „Reader“ und „Writer“). Konnektoren beinhalten mit Kommunikationsaspekten aus allgemeiner Sicht ebenfalls Verarbeitungsschritte (z. B. Protokollverarbeitung oder Verteilung). In einigen ADLs werden Konnektoren deshalb auch nicht explizit modelliert [MDE+95, LuV95, Ves96]. Nach [Sha93] stellen Konnektoren aber eine wesentliche Abstraktion für die Kommunikation und Interaktion zur Verfügung, die eine separate Beschreibung rechtfertigt. So werden diese Aspekte aus den Komponenten in Konnektoren verlagert und können auf diese Weise unabhängig beschrieben werden. Komponenten werden damit unabhängig von bestimmten Interaktionstypen und –protokollen. Die Übersetzung von Konnektoren resultiert anders als bei Komponenten nicht notwendigerweise in einer Ausführungseinheit, sondern diese können gegebenenfalls auch in entsprechende Datenstrukturen, Puffer oder Sequenzen von Anweisungen und Prozeduraufrufen im Code abgebildet werden [MeT00].

Die Beziehungen zwischen Komponenten und Konnektoren werden durch die Konfiguration definiert. Konfigurationen können hierarchisch beschrieben werden, d. h. Komponenten und Konnektoren müssen keine atomaren Elemente sein, sondern können selbst durch

eine Konfiguration erzeugt werden. Konfigurationen sind damit hierarchische Graphen von Komponenten als Knoten und Konnektoren als Kanten. Die Konfigurationsbeschreibung einer Hierarchieebene wird Repräsentation genannt. Representation Maps definieren Zusammenhänge zwischen der internen Struktur und den externen Schnittstellen von Konfigurationen auf benachbarten Hierarchieebenen. Im einfachsten Fall können interne Ports verschiedener Komponenten als externe Ports einer zusammengesetzten Komponente definiert werden. Es sind jedoch auch komplexere Abbildungen möglich.

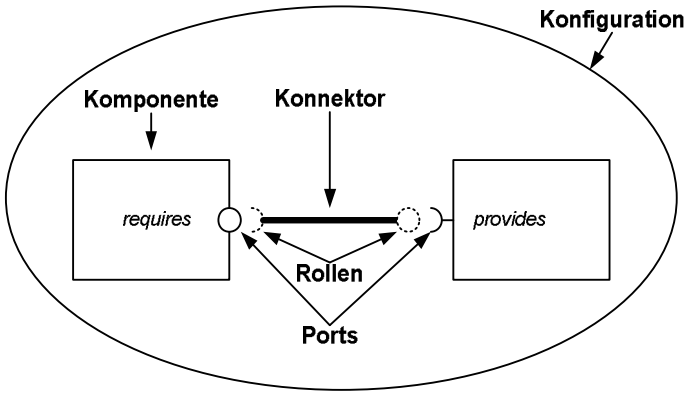


Abbildung 2-17: Elemente der Strukturbeschreibung von ACME

Neben der Strukturdefinition können in vielen ADLs weitere architekturspezifische Eigenschaften definiert werden (z. B. Typinformationen, Laufzeiteigenschaften, Protokollinformationen, Ressourcenverbrauch etc.). In ACME werden diese mit Hilfe von Properties abgebildet. Jedem der sieben Grundelemente können Properties zugewiesen werden. Eine Propertydefinition besteht aus einem Namen, einem optionalen Typen und einem Wert. Der Typ eines Properties identifiziert eine Untersprache zur Beschreibung des zugehörigen Property-Wertes. In ACME selbst werden nur einfache Typen wie Boolean, Integer oder String definiert. Aus Sicht von ACME werden Property-Informationen nur beschrieben, eine Interpretation erfolgt nur durch entsprechende Werkzeuge. Damit können erweiterte Informationen in die Architekturbeschreibung eingefügt werden. Neue Informationen und Untersprachen können einfach hinzugefügt werden, ohne die Architekturbeschreibungssprache ändern zu müssen. Die Definition erweiterter Eigenschaften bleibt jedoch auf den Property-Mechanismus beschränkt.

Zur Definition wiederkehrender Muster in Architekturen enthält ACME einen Template-Mechanismus. Templates entsprechen parametrisierbaren Makros und beschreiben Hüllen von Grundelementen, denen Parameter übergeben werden können, um konkrete Instanzen zu erzeugen. Templates können sowohl für Komponenten und Konnektoren als auch für Konfigurationen definiert werden. Style-Definitionen erlauben außerdem die Gruppierung von zusammengehörigen Templates, z. B. Templates für Client- und Serverkomponenten sowie einen RPC-Konnektor zur Beschreibung einer Client/Server Architektur.

Neben diesem Kern von Elementen zur Architekturbeschreibung enthalten ADLs weitere Elemente zur Modellierung spezifischer Aspekte von Softwarearchitekturen. Diese Elemente sind im Allgemeinen mit einer Werkzeugunterstützung gekoppelt. Das Spektrum der untersuchten Aspekte sowie der verfügbaren Werkzeuge ist sehr breit. So ist Rapide [LuV95] eine allgemein verwendbare Sprache zur Beschreibung von Systemen. Dagegen zielen Darwin [MDE+95] und C2 [MOR+96] auf die Beschreibung stark verteilter und dynamischer Systeme. Wright [AlG97] untersucht die Modellierung und Analyse konkurrenter Systeme und Unicon [SDK+95] betrachtet die Generierung von Code zur Verbin-

dung von Komponenten über verschiedene Interaktionsprotokolle. Verfügbare Werkzeuge unterstützen die Visualisierung, Analyse und Compilierung bis hin zur Codeerzeugung.

Nicht zuletzt die aus den unterschiedlichen Ausrichtungen resultierende Vielfalt an Sprachkonstrukten, der heterogene Formalisierungsgrad und der geringe Konsens bezüglich dieser Aspekte haben die Verbreitung und den Einsatz von ADLs in der Praxis bisher verhindert. Insbesondere existiert kein Standard einer Architekturbeschreibungssprache. ACME als Austauschsprache für Architekturbeschreibungen stellt den kleinsten gemeinsamen Nenner von ADLs dar, die spezifischen Eigenschaften der einzelnen ADLs werden in Form von Properties in der jeweiligen Sprache formuliert. Elemente aus dem Sprachkern, die der Beschreibung von Architekturen dienen wurden aber zum Teil in UML 2.0 integriert [OMG03a]. So sind neben Komponenten auch Ports und Konnektoren Teil des UML-MetaModells. Weitere Versuche der Integration von ADLs in UML sind z. B. die Definition von ACME als UML Profile auf Basis von UML 2.0 [GoA03] und die Definition von C2 und Wright mit UML 1.3 [MRR+00]. Eine zusammenfassende Betrachtung und eine Klassifikationsschema für ADLs ist unter [MeT00] zu finden.

2.5.5 Unified Modeling Language

Die Unified Modeling Language (UML) ist ein Standard zur Visualisierung, Spezifizikation, Konstruktion und Dokumentation von Softwaresystemen [OMG03c]. UML orientiert sich am objekt-orientierten Softwareentwurf und bietet im Wesentlichen eine einheitliche Notation und Semantik sowie die Definition eines Meta-Modells. Die Modellierungssprache ist seit 1997 ein Standard der Object Management Group, die aktuelle Version des Standards ist UML Version 1.5 [OMG03c].

Das UML-Modell eines Softwaresystems besteht aus mehreren Teilmodellen, die jeweils eine bestimmte Sicht auf das System beschreiben. Dazu steht eine Menge von Diagrammtypen zur Verfügung, die eine Modellierung unterschiedlicher Abstraktionsebenen und Detaillierungsgrade ermöglichen und gemeinsam das Gesamtsystem beschreiben.

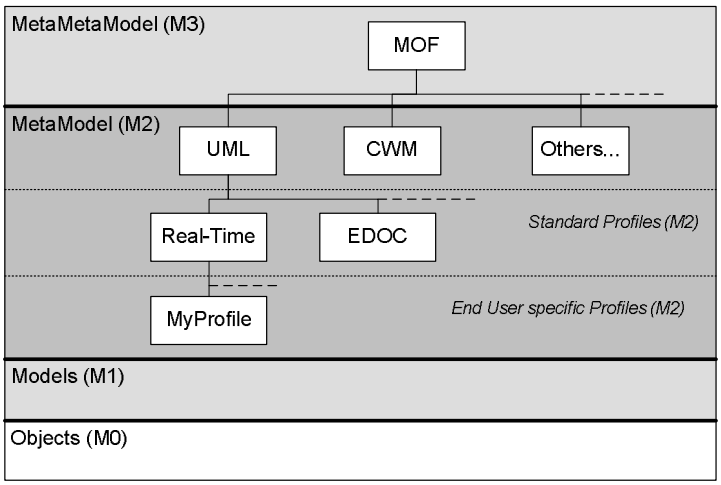


Abbildung 2-18: Spezialisierung von Standardmodellen durch Profiles (nach [OMG99a])

UML ist eine Definition innerhalb einer Vier-Schichten-Architektur für Modelle, die von der OMG standardisiert wurde (siehe Abbildung 2-18). Die Ebene des Meta-Meta-Modells (M3) wird durch den Standard Meta-Object Facility (MOF) [OMG02c] definiert und ist die Grundlage zur Definition und Standardisierung verschiedener Meta-Modelle (M2). Eines der durch MOF definierten Meta-Modelle ist UML. Modelle von Softwaresystemen wer-

den in UML auf der Ebene M1 definiert. Auf der Schicht M0 befinden sich die konkreten Objekte, die Modellelemente aus dem UML Modell instantiieren. UML Modelle sind somit Instanzen der Meta-Klassen, die Elemente des UML Meta-Modells sind.

2.5.5.1 Meta-Object Facility

Der MOF Standard (Meta-Object Facility) definiert eine abstrakte Sprache zur Spezifikation von Meta-Modellen. Beispiele von existierenden Meta-Modellen auf Basis von MOF sind UML, CWM [OMG03b] und MOF selbst. Außerdem unterstützt MOF die Abbildung von Meta-Modellen auf Programmierschnittstellen für die Verarbeitung der durch das jeweilige Meta-Modell definierten Meta-Daten. MOF und UML besitzen eine gemeinsame Untermenge von Modellelementen, die Struktur beider Sprachen ist sehr ähnlich. Beide Sprachen sind derzeit jedoch noch unabhängig voneinander. Als Teil der EDOC Spezifikation wir deshalb ein UML Profile für MOF definiert [OMG04], das es ermöglicht, die graphische Notation sowie Werkzeuge für UML auch für die Spezifikation von MOF Meta-Modellen zu verwenden. Eine Vereinigung beider Sprachen ist für die Zukunft vorgesehen.

MOF unterstützt eine Teilmenge der Konstrukte von UML. Zum Teil besitzen diese die gleichen Eigenschaften, einige der Elemente unterscheiden sich jedoch im Detail. Dies ist vor allem in der unterschiedlichen Fokussierung beider Sprachen begründet. Während UML eine allgemein verwendbare Sprache zur Modellierung verteilter Objektsysteme ist, dient MOF vorrangig der Spezifikation von Meta-Modellen und beschreibt damit insbesondere keine Implementierungsaspekte. Wesentliche Eigenschaften von MOF sind die Objektorientierung, und die Ebenenstruktur, die nicht auf die in Abbildung 2-18 dargestellten vier Ebenen beschränkt ist. Außerdem wird MOF formal durch die eigenen Konstrukte definiert (siehe [OMG02c]).

MOF enthält vier wesentliche Modellierungskonzepte. Dies sind Klassen, Assoziationen, Datentypen und Packages. Die Klassen dienen der Modellierung der Meta-Objekte. Assoziationen beschreiben Beziehungen zwischen diesen Meta-Objekten. Datentypen ermöglichen die Beschreibung zur Modellierung gehörender Daten. Packages unterstützen die Modularisierung von Modellen. Wesentliche Unterschiede zwischen UML und MOF Elementen sind die Beschränkung auf binäre Assoziationen in MOF, ein zusätzliches Konzept von Referenzen in MOF und die Modellierung von Konzepten wie Generalisierung, Abhängigkeiten und Verfeinerungen in Form von Assoziationen in MOF im Gegensatz zu Klassen in UML. Einen detaillierten Überblick über die Unterschiede bzw. die Möglichkeiten zur Abbildung der Modellelemente zwischen UML und MOF enthält [OMG02c].

2.5.5.2 Erweiterungsmechanismen von UML

Das UML Meta-Modell wurde mit dem Ziel der Allgemeinheit definiert und ist für ein Spektrum von Anwendungsfällen einsetzbar. Eine Spezialisierung dieser allgemeinen Modellierungssprache ist für Domänen wünschenswert, um eine spezifische Notation und Semantik von Elementen verwenden zu können und damit die Ausdrucksstärke des Modells zu erhöhen.

Dazu kann UML durch zwei Mechanismen erweitert werden:

1. Leichtgewichtige Erweiterungen durch den in UML enthaltenen Erweiterungsmechanismus
2. Schwergewichtige Erweiterungen durch die Erweiterung des Meta-Modells auf MOF-Ebene (z. B. durch die Definition neuer Meta-Klassen)

Erweiterungen entsprechend des zweiten Mechanismus' verändern das UML Meta-Modell und sind damit nicht mehr konform zu UML. Insbesondere werden solche Erweiterungen nicht von Standard UML-Werkzeugen unterstützt. Erweiterungen durch den Standarderweiterungsmechanismus von UML verändern dagegen das UML Meta-Modell nicht und resultieren deshalb in Modellen die konform zum Standard UML Meta-Modell sind. Elemente des leichtgewichtigen Erweiterungsmechanismus sind: *Stereotypen, Tag Definitionen, Tagged Values* und *Constraints*.

Stereotypen

Ein Stereotyp ist ein Modellelement, das eine Meta-Klasse durch zusätzliche Tags, Contraints und optional eine neue graphische Notation erweitert. Stereotypen sind damit Unterklassen existierender Elemente des Meta-Modells und können als (virtuelle) Meta-Klassen betrachtet werden. Sie besitzen die gleiche Struktur wie ihre Meta-Klasse (z. B. Attribute und Beziehungen) sowie die zusätzlich definierten Elemente und eine erweiterte Semantik. Von Werkzeugen müssen sie deshalb insbesondere bei der Codeerzeugung in einer spezifischen Form behandelt werden. Es können drei Arten von Stereotypen unterschieden werden:

1. *Dekorative Stereotypen* definieren spezifische Notationen für Meta-Klassen.
2. *Deskriptive Stereotypen* beschreiben Verwendungszusammenhänge bzw. weisen Elementen Kommentare zu, die sie von der Meta-Klasse unterscheiden.
3. *Restriktive Stereotypen* definieren formale Einschränkungen auf das Vorhandensein bzw. Nicht-Vorhandensein bestimmter Eigenschaften in Bezug auf die Meta-Klasse.

Einem Stereotyp können Tagged Values und Contraints zugeordnet werden.

Tag Definitionen und Tagged Values

Tagged Values sind Eigenschaftswerte, die Modellelementen zugewiesen werden können. Diese besitzen ein Schlüsselwort und einen oder mehrere Werte, die einem bestimmten Typ entsprechen müssen. Diese Typdefinition wird durch eine Tag Definition realisiert. Neben dem Typen eines Tagged Values legt die Definition auch die minimale und maximale Anzahl der Werte fest, die einem Tagged Value zugewiesen werden können. Wert eines Tagged Values kann auch eine Referenz auf ein weiteres Modellelement sein. Seit UML 1.4 sind Tag Definitionen an Stereotypen gebunden. Damit können Tagged Values nur noch Modellelementen zugeordnet werden, die mit einem Stereotypen versehen sind, dem eine Tag Definition zugeordnet ist.

Constraints

Constraints formulieren Einschränkungen oder Bedingungen für Inhalte, Zustände oder die Semantik von Modellelementen und müssen stets erfüllt sein. Werden Constriants Stereotypen zugewiesen, schränken sie die Semantik in Bezug auf die Meta-Klasse weiter ein

bzw. spezialisieren diese. Formal definierte Constraints können durch Werkzeuge ausgewertet und überprüft werden.

Durch die beschriebenen Elemente sind nur additive Änderungen des Meta-Modells möglich. Elemente des Meta-Modells können also nicht explizit entfernt werden. Von UML durch die Standardmechanismen zur Erweiterung abgeleitete Modelle sind weiterhin konform zum UML Standard und können in Standardwerkzeugen genutzt werden. Insbesondere kann jedes Modell, das von einem Profil abgeleitet wurde, auch aus der Perspektive des übergeordneten Meta-Modells betrachtet werden.

Eine Menge von zusammengehörenden Erweiterungselementen kann zu einem UML Profil zusammengefasst werden. Profile enthalten Stereotypen, Tagged Values, Constraints und Datentypen zur Modellierung einer spezifischen Anwendungsdomäne. Standardisierte Profile sind z. B. das UML Profile für Echtzeit Anwendungen (Realtime) und für verteilte, objekt-orientierte Unternehmensanwendungen (Enterprise Distributed Object Computing, EDOC). Profile für Implementierungstechnologien sind die Profile für CORBA [OMG02d] und EJB [Gre01]. Weitere Profile können entweder aus UML aber auch aus UML Profilen abgeleitet werden (siehe Abbildung 2-18).

2.5.6 Model Driven Architecture

Während ihrer Lebenszeit durchläuft Software einen Entwicklungsprozess und unterliegt einer ständigen Weiterentwicklung und Veränderung. Während dieses Prozesses wechseln die daran beteiligten Personen ebenso wie die zugrunde liegenden Plattformen und Technologien (Programmiersprachen, Middleware, Betriebssystem, Hardware, usw.). Die UML unterstützt die Anwendungsentwicklung mit vornehmlich grafischen Modellen, die sowohl die Struktur als auch das Verhalten von Software beschreiben können. Jedoch vollzieht sich mit dem Übergang vom Design zur Implementierung ein Wechsel der Repräsentation von Entwicklungsmodellen zu Quelltext und gleichzeitig auch der Werkzeugumgebung. Damit werden die Entwicklungsmodelle weitgehend von der weiteren Entwicklung am Quellcode entkoppelt. Eine Wahrung der Konsistenz zwischen beiden Modellwelten stellt dadurch einen erheblichen Aufwand dar. Dies führt dazu, das UML in vielen Entwicklungsprojekten gar nicht oder nur in einem frühen Stadium zur Visualisierung bzw. nachträglich zur Dokumentation eingesetzt wird, am eigentlichen Entwicklungsprozess aber nur selten beteiligt ist [StB03].

Die Model Driven Architecture (MDA) [MiM03] der OMG stellt das Modell in den Mittelpunkt des Softwareentwicklungsprozesses. Ein Modell wird im Kontext von MDA als Abstraktion verstanden, das durch eine Anreicherung mit Informationen über die Ausführungsplattform schrittweise verfeinert und dadurch in ein (nahezu) lauffähiges Softwaresystem überführt werden kann. Damit soll eine durchgängig modellbasierte Softwareentwicklung und die Automatisierung eines Teils der Entwurfsschritte möglich werden. Zum Erreichen dieser Zielstellung verfolgt MDA den Ansatz der Trennung der Spezifikation eines Systems von den Details der Plattform, auf der das System realisiert wird. Dazu werden in [MiM03] drei verschiedene Perspektiven (Viewpoints) auf ein System definiert, die den Fokus der Betrachtung auf bestimmte Konzepte und Strukturregeln eines Systems legen, um diese zu betonen. Diese drei Perspektiven sind:

1. Computation Independent Viewpoint (CIM): Diese Perspektive modelliert ein System unabhängig von der Struktur sowie Verarbeitungsaspekten des Systems. Damit können vor allem Anforderungen und Informationen zur Anwendungsdomäne des Systems erfasst werden.

2. Platform Independent Viewpoint: Diese Perspektive erfasst die formale Spezifikation und Funktionalität eines Systems, abstrahiert aber von den spezifischen Details von Plattformen.
3. Platform Specific Viewpoint: Diese Perspektive fügt der Systemspezifikation Details zur Realisierung des Systems auf einer spezifischen Plattform hinzu.

Eine Sicht modelliert ein System aus einer bestimmten Perspektive. Dem entsprechend können nach [MiM03] die Sichten Computation Independent Model (CIM), Platform Independent Model (PIM) und Platform Specific Model (PSM) unterschieden werden. MDA definiert ein Muster, nach dem ein plattformunabhängiges Modell eines Systems in dessen plattformspezifisches Modell transformiert werden kann. In die Transformation fließen weitere Informationen, beispielsweise über die Plattform in Form einer Plattform Spezifikation, mit ein (siehe Abbildung 2-19).

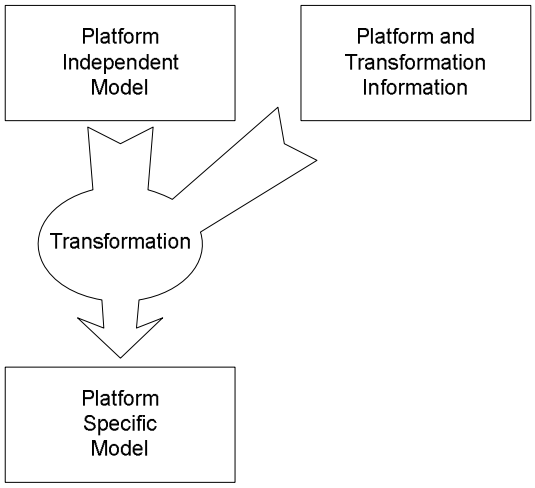


Abbildung 2-19: MDA Muster zur Modell-Transfrmation (nach [MiM03])

Die Transformationsregeln basieren auf Typinformationen der beiden Modelle (Model Type Mapping), plattformspezifischen Markierungen der Elemente des PIM sowie auf Elementen der Meta-Modelle von PIM und PSM. Das Ergebnis der Transformation ist ein plattformspezifisches Modell des durch das PIM beschriebenen Systems. Durch die Transformation wurden Informationen über die Beziehung des Systems zur Plattform hinzugefügt. Diese Beziehungen können mehr oder weniger detailliert beschrieben werden. Enthält das PSM alle Informationen zur Erzeugung und Ausführung des Systems innerhalb der Plattform, beschreibt das PSM eine Implementierung. Andernfalls kann das PSM erneut als PIM betrachtet und weiter verfeinert werden. Das MDA Muster zur Modelltransformation kann also mehrfach auf Modelle eines Systems angewendet werden.

Ziel des MDA Ansatzes ist also eine möglichst durchgängige Unterstützung des Softwareentwurfs durch Modelle. Diese liegen insbesondere in maschinenlesbarer Form vor und sollen eine Automatisierung von Teilaufgaben des Softwareentwurfs ermöglichen. Die Trennung der Systemspezifikation von Plattforminformationen ermöglicht insbesondere die plattformunabhängige Beschreibung von Softwaresystemen und damit eine leichte Portierbarkeit, Wiederverwendbarkeit sowie Interoperabilität der beschriebenen Systeme. Diese Eigenschaften tragen insbesondere der schnellen Entwicklung der Technologien Rechnung, von denen die Systemmodelle zum großen Teil unabhängig werden.

2.5.7 Aspektorientierte Softwareentwicklung

Die gegenwärtige Softwareentwicklung wird von dem Paradigma der Objekt-Orientierung dominiert. Dies spiegelt sich in den Methodologien, Analyse- und Entwicklungswerkzeugen sowie den Programmiersprachen wider. Die Idee dieses Paradigmas ist die Zerlegung von Softwaresystemen bzw. Problemen in Objekte. Diese stellen autonome Einheiten dar, die Code und Daten für ein Teilproblem einkapseln. Damit wird eine Modularisierung von Software auf der Ebene der Funktionalität erreicht. Dieses Konzept lässt sich gut auf Problemstellungen anwenden, die sich in Teilprobleme zerlegen lassen, welche innerhalb eines Objektes gelöst werden können. Die Objektorientierung stößt jedoch an ihre Grenzen, wenn sich Problembereiche überschneiden bzw. nicht einem bestimmten Objekt zugeordnet werden können, sondern vielmehr verteilt auf verschiedene Objekte realisiert werden müssen.

Das Problem sich überschneidender Problembereiche (crosscutting concerns) wird von dem Ansatz der aspektorientierten Softwareentwicklung (Aspect-Oriented Software Development, AOSD) [EFB01, EAK+01] adressiert. Der Ansatz des AOSD basiert auf einer getrennten Spezifikation sich überschneidender Problembereiche sowie deren Beziehungen untereinander. Diese separaten Beschreibungen werden dann durch Werkzeuge zu einem Gesamtsystem „verwebt“. Problembereiche bzw. Aspekte in AOSD können abstrakt (z. B. Sicherheit oder Quality of Service) aber auch konkret sein (z. B. Caching, Pufferung). Außerdem können Aspekte sowohl funktional (Geschäftsregeln) als auch nicht-funktional (Synchronisation, Transaktionsmanagement) sein. AOSD betrachtet diese Aspekte als Elemente „erster Klasse“. Aspekte können auf allen Ebenen des Lebenszyklus’ von Software betrachtet werden.

Die aspektorientierte Softwareentwicklung ersetzt keine Technologien, sondern ergänzt bzw. erweitert diese. Aspekte werden von der Kernfunktionalität eines Systems getrennt und durch einen expliziten Mechanismus zu einem Gesamtsystem integriert. Dabei erfolgt eine Vermischung sowohl mehrerer Aspekte in Funktionsmodulen als auch eine Verteilung einzelner Aspekte über mehrere Funktionsmodule. Die wesentlichen Elemente der aspektorientierten Softwareentwicklung sind die Beschreibung der Aspekte, der Punkte, an denen Aspektcode eingefügt werden soll (Join Points) sowie der Mechanismus zur Integration der Aspekte in das Gesamtsystem (Aspect Weaver) (siehe z. B. [OsT00, KHH+01]).

Das Ziel der aspektorientierten Softwareentwicklung ist eine Erweiterung der Modularisierung gegenüber einer objekt-orientierten Betrachtung. Sich durchdringende Problembereiche werden aus dem Gesamtsystem herausgetrennt. Danach kann eine separate Betrachtung und Beschreibung der einzelnen Aspekte erfolgen. Die Integration von Aspektverhalten wird durch Join Points beschrieben, die Anknüpfungspunkte an die Kernfunktionalität eines Systems darstellen. Realisiert wird die Integration durch einen Webemechanismus. Während bei MDA die Transformation zwischen unterschiedlich abstrakten Modellen in vertikaler Ebene betrachtet wird, wird bei AOSD ein Gesamtsystem auf einer Ebene horizontal in Aspekte zerlegt. Damit betrachtet MDA ein Gesamtsystem in verschiedenen Ebenen, wogegen der Focus bei AOSD auf der Spezifikation der einzelnen Mechanismen sowie der Webepunkte auf einer Ebene liegt.

2.5.8 Zusammenfassung

Die im fünften Teil dieses Kapitels beschriebenen Lösungsansätze stellen allgemeine Ansätze zur Softwareentwicklung und insbesondere zur Architekturbeschreibung dar. Das Architekturmuster „Pipes and Filters“ liegt vielen der Lösungen aus Teil 2 in zum Teil

erweiterter Form zugrunde und stellt somit eine wesentliche Lösungsidee für eine flexible und lose Kopplung autonomer Adaptionsmechanismen dar. Softwarekomponenten sind ebenfalls unabhängige Module, die miteinander kombiniert werden können. Insbesondere wird durch die explizite Beschreibung der Schnittstellen sowie weiterer Kontextabhängigkeiten eine hohe Wiederverwendbarkeit angestrebt. Softwarekomponenten können für die Beschreibung von Filtern verwendet werden und spielen auch in den weiteren Ansätzen eine wesentliche Rolle.

Der Ansatz der Konfigurationsprogrammierung stellt eine grundsätzliche Lösungsmöglichkeit zur Beschreibung von Architekturen dar. Betrachtet werden verteilte Systeme im Allgemeinen. Es erfolgt explizit eine Trennung der Beschreibung von Einzelkomponenten sowie der Kommunikation und Struktur der Anwendung. Auf dieser Grundlage können Adaptionsmechanismen als Komponenten beschrieben und flexibel zu Anwendungen kombiniert werden. Durch eine getrennte Betrachtung genutzter und bereitgestellter Schnittstellen wird der Forderung nach ausschließlich expliziten Kontextabhängigkeiten entsprochen. Dies führt zu einer erhöhten Autonomie der Komponenten. In ähnlicher Weise unterstützen Architekturbeschreibungssprachen die Spezifikation verteilter Anwendungen. Wesentliche Elemente der Konfigurationsprogrammierung sind auch bei diesen Sprachen zu finden. Außerdem wird durch Konnektoren eine explizite Beschreibung von Aspekten der Kommunikation innerhalb von Architekturen möglich, die damit aus den Komponenten in separate Elemente verlagert werden. Dies führt zu einer weiteren Unabhängigkeit der Komponenten. Beide Ansätze sehen auch eine hierarchische Beschreibung von Anwendungen vor, d. h. Komponenten werden zu umfangreicheren Komponenten zusammengesetzt und bilden letztendlich die Anwendung. Im Gegensatz zur Konfigurationsprogrammierung sind eine Laufzeitunterstützung sowie Tools zum Management der Anwendungen zur Laufzeit aber zumeist keine Schwerpunkte von ADLs.

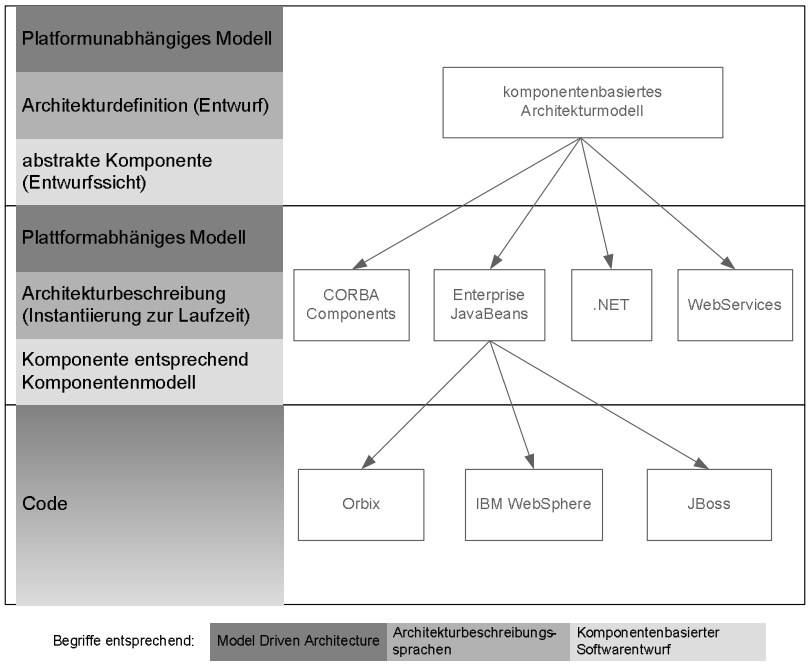


Abbildung 2-20: Modellstufen für MDA, Architekturbeschreibungssprachen und komponentenbasierte Softwareentwicklung

UML ist die gegenwärtige Standardsprache zur Softwareentwicklung. Architekturbeschreibungen werden im Vergleich zu ADLs und der Konfigurationsprogrammierung jedoch weit weniger unterstützt. Mit Hilfe des spracheigenen Erweiterungskonzeptes kön-

nen aber mit UML Profiles Erweiterungen vorgenommen werden. Außerdem bietet UML 2.0 neue Konzepte für die Beschreibung von Architekturen, wie die explizite Definition von „provides“ und „requires“ Schnittstellen und die Unterstützung zusammengesetzter Komponenten. Die Ansätze der Model Driven Architecture sowie der aspektorientierten Programmierung stellen neue Konzepte der Softwareentwicklung dar. Beide Ansätze bieten auch in Bezug zur Adaption neue Konzepte. Die aspektorientierte Programmierung setzt den Schwerpunkt auf einzelne Mechanismen, die als Aspekte separat beschrieben werden. Die Model Driven Architekture betrachtet dagegen Modelle einer vollständigen Anwendung in verschiedenen Abstraktionsebenen und betont damit die Zusammenhänge im Gesamtsystem. Zwischen den Architekturebenen der MDA und unterschiedlichen abstrakten Sichten in komponentenbasierten Systemen und ADLs können Zusammenhänge hergestellt werden (siehe Abbildung 2-20). Insbesondere die Perspektive der Architektur unabhängig von einer Implementierungsplattform soll in der Arbeit aufgegriffen werden, um adaptive Anwendungen zu beschreiben.

2.6 Fazit

Die Betrachtungen zu Beginn des Kapitels haben gezeigt, dass bereits zahlreiche Projekte existieren, die bestimmte Teilprobleme der Adaption adressieren. Es sind also eine Vielzahl von Mechanismen verfügbar, die eine Lösung der in Kapitel 1 definierten Anforderungen A1.1 bis A1.4 ermöglichen. Der zweite Teil des Kapitels zeigt weiterhin, dass diese Mechanismen in Form von Komponenten miteinander kombiniert werden können. Die beschriebenen Lösungen konzentrieren sich aber vorwiegend auf die Suche von Mechanismen zur Erzeugung von Pfaden bzw. Graphen und auf Laufzeitmechanismen. Die Definition wieder verwendbarer Komponenten steht im Hintergrund. Außerdem betrachtet die Mehrzahl der Arbeiten im ersten und zweiten Teil Adaption beschränkt auf spezifische Anwendungsdomänen oder auf eine Teilmenge von Mechanismen. Insbesondere besteht kein Bezug zur Softwaretechnologie.

Eine Systematisierung in Form einer Klassifikation der vorhandenen Mechanismen sowie eine Betrachtung zu deren Anwendbarkeit wurde nur in wenigen Arbeiten thematisiert. Die existierenden Klassifikationen werden in Kapitel 3 detailliert beschrieben. Außerdem wird aus den in diesem Kapitel gewonnenen Erkenntnissen ein Klassifikationsschema der Adaptionsmechanismen in Bezug auf die in Kapitel 1 genannten Anforderungen entwickelt.

Neben der Systematisierung und Einordnung der vorhandenen Adaptionsmechanismen fehlt auch eine Unterstützung der Beschreibung und Entwicklung adaptiver Anwendungen. In den Arbeiten in Teil 1 und Teil 2 wurden vorwiegend Prototypen von Anwendungen und Laufzeitsystemen entworfen. Damit stehen Erkenntnisse über die notwendige Unterstützung der Adaption zur Laufzeit zur Verfügung. Einige generische Aspekte werden in Teil 3 beschrieben, Standards für Middleware-Plattformen für mobile Infrastrukturen fehlen jedoch bisher. Die in Teil 5 beschriebenen Ansätze zur Softwareentwicklung sind ebenso allgemein und bieten keine dedizierte Unterstützung für adaptive Anwendungen. Die beschriebenen Sprachen und Konzepte bieten aber eine gute Grundlage für Erweiterungen in Hinblick auf eine Entwicklungsunterstützung für adaptive Anwendungen. Die Model Driven Architecture stellt die modellbasierte Entwicklung von Anwendungen in den Mittelpunkt. Insbesondere die Spezifikation plattformunabhängiger Modelle erlaubt eine allgemeine Sicht auf bestimmte Eigenschaften einer Anwendung (z. B. die Adaption). Architekturbeschreibungssprachen sowie Ansätze zum Konfigurationsmanagement beschreiben Anwendungen in einer solchen abstrakten Sicht mit dem Schwerpunkt einer Architekturbeschreibung. Diese Sicht ermöglicht die Betrachtung von Strukturänderungen zur Adaption. Die getrennte Betrachtung von bereit-

zur Adaption. Die getrennte Betrachtung von bereitgestellten und benötigten Schnittstellen einer Komponente ist dabei für die Wiederverwendbarkeit und flexible Verbindung von Komponenten wesentlich.

Im vierten Teil wird Kontext als Information zur Steuerung von Adaptionsprozessen untersucht. Im Forschungsbereich „Context-Awareness“ wurden bereits weitreichende Betrachtungen zur Gewinnung von Kontext sowie zur Realisierung eines Kontextdienstes durchgeführt. Die Verwendung von Kontext erfolgte jedoch überwiegend in Anwendungen, die nicht auf die Adaption im Sinne der Arbeit zielen. Die im ersten Teil beschriebenen Lösungen setzen Kontextinformationen häufig implizit zur Steuerung der Adaption voraus, nur wenige Arbeiten betrachten aber explizit den Zusammenhang zwischen Kontextabhängigkeit und Adaption.

Die durch die eingehende Analyse des State-of-the-Art gewonnenen Erkenntnisse sollen nachfolgend zur Identifizierung wiederverwendbarer Basismechanismen zur Adaption verwendet werden. Außerdem haben die Betrachtungen in Abschnitt 2.4 gezeigt, das Kontext die Informationen über die Ausführungsumgebung bereitstellt, die für adaptive Anwendungen in mobilen verteilten Infrastrukturen von wesentlicher Bedeutung sind. Dies sind technische Informationen über das verwendete Zugangsnetzwerk sowie Informationen über die Ressourcen des Endgerätes. Außerdem sind Informationen über den Anwender und die jeweilige Anwendungssituation verfügbar. In Abschnitt 2.1.3.5 wird außerdem der Einsatz von manuell erzeugten Meta-Informationen zur Verbesserung der Resultate automatischer Adaptionsprozesse beschrieben. Diese Erkenntnisse dienen gemeinsam mit den identifizierten und klassifizierten Basismechanismen als Grundlage zur Erarbeitung eines Meta-Modells zur Beschreibung adaptiver Anwendungen entsprechend der Zielstellung der vorliegenden Arbeit.

Kapitel 3

KLASSIFIKATION UND BEWERTUNG VON BASISMECHANISMEN ZUR ADAPTION

In den ersten beiden Teilen des Kapitels 2 wurde eine Vielzahl von Forschungsarbeiten beschrieben und diskutiert, die Lösungsansätze zur Adaption in heterogenen und dynamischen und insbesondere auch mobilen Ausführungsumgebungen untersuchen. In der Beschreibung wurden die jeweils verwendeten Adaptionsmechanismen herausgearbeitet. Bei einem Vergleich der Lösungsansätze lassen sich Gemeinsamkeiten hinsichtlich der eingesetzten Mechanismen erkennen. Ein Beispiel ist der Mechanismus Caching, der zur Realisierung abgekoppelter Operationen (Abschnitt 2.1.1.1), zur Reduzierung der Antwortzeit in Browsern, ebenso wie zur Zwischenspeicherung von Paketen im TCP-Snoop-Protokoll verwendet wird (Abschnitt 2.1.2.1).

In diesem Kapitel sollen grundlegende Mechanismen zur Unterstützung heterogener und dynamischer Ausführungsumgebungen identifiziert, klassifiziert und entsprechend der Anforderungen A1.1 bis A1.4 aus Kapitel 1 bewertet werden. Dazu wird ein Klassifikationsmodell entwickelt, das eine systematische Einordnung von Basismechanismen zur Adaption ermöglicht. Die in Kapitel 2 beschriebenen Adaptionsmechanismen werden in das entwickelte Schema eingeordnet. Da die Benennung vieler Adaptionsmechanismen in der Literatur anhand von Begriffen erfolgt, die sich überschneidende Klassen von Mechanismen bezeichnen bzw. in Bezug auf die Klassen des entwickelten Schemas nicht eindeutig sind, erfolgt eine Begriffsklärung bezüglich der Benennung der Adaptionsmechanismen im Rahmen der Arbeit. Abschließend werden auf Basis der Anforderungen A1.1 bis A1.4 aus Kapitel 1 Kriterien für die Bewertung erarbeitet, die auf die identifizierten Adaptionsmechanismen angewendet werden.

3.1 Ein Klassifikationsschema für Basismechanismen zur Adaption

Die Ausführungen in Kapitel 2 haben gezeigt, dass vielfältige Mechanismen eingesetzt werden können, um Anwendungen an sich dynamisch ändernde Ausführungsumgebungen anzupassen. Die eingesetzten Mechanismen sind überwiegend bekannt und nur zum Teil selbst adaptiv, in dem Sinne, dass sie Ergebnisse innerhalb eines bestimmten Wertebereichs liefern können. Viele der Mechanismen wurden für eine bestimmte Ausführungsumgebung spezialisiert (z. B. verschiedene TCP-Protokolle). Ein adaptiver Mechanismus ist z. B. die Skalierung eines Bildes, die aus einem Originalbild innerhalb eines bestimmten Bereiches verschieden große Zielbilder erzeugen kann. Im Gegensatz dazu kann etwa das Verfahren zur Kompression der Paketköpfe für TCP-Verbindungen entweder eingesetzt werden oder nicht (Abschnitt 2.1.2.2). Ein adaptives Verhalten wird meist erst durch die Kombination von Mechanismen, eine gezielte Platzierung, eine dynamische Konfiguration

und/oder das dynamische Ersetzen verschiedener spezialisierter Mechanismen in Abhängigkeit von der Ausführungsumgebung erreicht.

Eine Systematisierung und Einordnung der beschriebenen Mechanismen soll einerseits einen Überblick über die verfügbaren Mechanismen und deren Anwendungsfelder geben und andererseits einen Vergleich verschiedener Ansätze ermöglichen.

3.1.1 Systembasierte vs. anwendungsbasierte Adaption

Eine erste Systematisierung von Ansätzen zur Adaption wird in [NoS95] und [Sat96] beschrieben. Demnach können Lösungen in einen Bereich eingeordnet werden, der durch zwei Extreme begrenzt wird (siehe Abbildung 3-1). Das eine Extrem ist die Realisierung adaptiver Anwendungen ausschließlich durch die Anwendung selbst (*laissez-faire*). Für diesen Ansatz ist keine Unterstützung durch das System notwendig oder vorgesehen. Die Anwendung muss selbständig Informationen über die Ausführungsumgebung beschaffen und steuert autonom die Verwendung von Ressourcen und die Operationen zur Adaption. Eine Optimierung der Ressourcenvergabe zwischen konkurrierenden Anwendungen und eine Koordination der Adaption über Anwendungsgrenzen hinweg ist damit nicht möglich.

Das zweite Extrem stellt die Realisierung der Adaption ausschließlich im System dar (*application-transparent*). Dies hat den Vorteil, dass Anwendungen nicht verändert werden müssen, um adaptiv zu werden. Außerdem kann eine zentrale Steuerung und damit eine optimale Ressourcenvergabe und Koordination der Adaption konkurrierender Anwendungen erfolgen. Ein wesentlicher Nachteil ist jedoch, dass Informationen zur Steuerung von Adaptionsoperationen nicht verwendet werden können, wenn diese nur auf Anwendungsebene verfügbar sind. Die vom System ausgeführten Adaptionsoperationen können somit inadäquat oder kontraproduktiv für die jeweilige Anwendung sein.

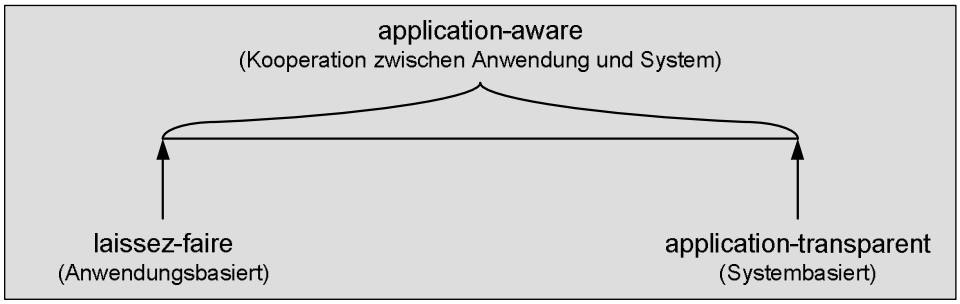


Abbildung 3-1: Bereich von Adaptionsstrategien (nach [Sat96])

Zwischen diesen beiden Extremen werden alle Ansätze eingeordnet, die eine Kooperation zwischen Anwendung und System anstreben (*application-aware*). Mit diesem Vorgehen sollen die Vorteile der beiden generellen Ansätze verbunden werden. Zum einen können so Informationen auf Anwendungsebene für die Steuerung der Adaption genutzt werden, zum anderen kann eine Koordination der Ressourcenvergabe und der Adaption über Anwendungsgrenzen hinweg durch das System erfolgen.

Entsprechend der vorgestellten Systematisierung können Lösungen wie das verteilte Dateisystem CODA als systembasiert eingeordnet werden. Die meisten der in der Literatur beschriebenen Ansätze sind jedoch Mischformen, d. h. sie beinhalten Mechanismen auf System- und Anwendungsebene (application-aware). Eine genauere Unterscheidung des größten Teils der existierenden Lösungen ist somit durch diese Klassifizierung nicht mög-

lich. Eine wesentliche Ursache dafür ist eine zu grob-granulare Betrachtung der Adaption. Adaptionsoperationen werden als Ganzes entweder dem System oder der Anwendung bzw. beiden Ebenen zugeordnet. Die Mehrzahl der Lösungen weist darauf hin, das durch die Einbeziehung aller relevanten Informationen sowohl auf System- (z. B. die verfügbare Datenrate oder die CPU-Auslastung) als auch auf Anwendungsebene (z. B. die Relevanz von Teildaten für die Gesamtinformation) meist bessere Lösungen gegenüber einer strikt abgegrenzten Betrachtung erreicht werden. Neben der Ressourcenverwaltung und Koordinationsmechanismen können auch Adaptionsoperationen selbst im System angesiedelt werden, wenn diese nicht von Anwendungsinformationen abhängen bzw. anwendungsübergreifend eingesetzt werden können (z. B. eine verlustfreie Kompression oder das Zwischenspeichern von Dateien in einem Cache).

3.1.2 Platzierung und Universalität

Die Klassifikation in [SpS00] strebt deshalb eine differenziertere Betrachtung der Adaption an. Adaption wird fein-granular anhand der verwebdeten Mechanismen unterschieden. Die Systematisierung folgt den Überlegungen, ob Mechanismen spezifisch für eine Anwendung sind oder von verschiedenen Anwendungen genutzt werden können (Universalität) und wo diese am effizientesten eingesetzt werden können (Platzierung). Demnach werden wie in Abbildung 3-2 dargestellt, die Mechanismen in vier Hauptklassen eingeteilt. Das vom Endgerät genutzte Zugriffsnetzwerk wird dafür als schwächstes Glied der Verbindung zwischen Endgerät und Diensten im Festnetz angenommen.

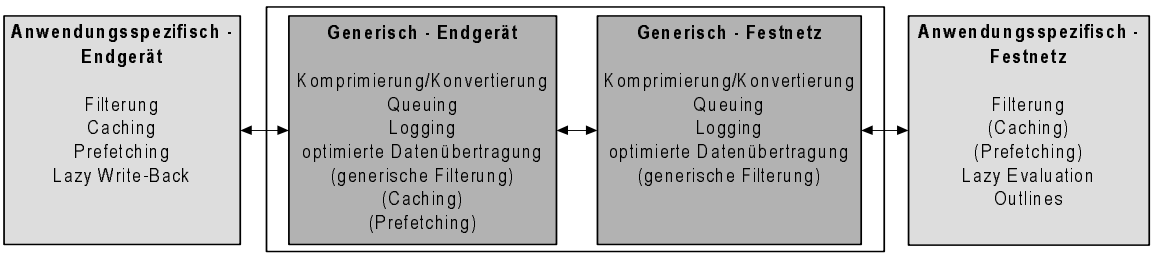


Abbildung 3-2: Klassifikation von Adaptionsmechanismen [SpS00]

Die Verfügbarkeit der Informationen zur Steuerung der Adaption spielt eine wesentliche Rolle für die Unterscheidung anwendungsspezifischer und generischer, d. h. anwendungsunabhängiger Mechanismen. So werden Mechanismen als anwendungsspezifisch eingeordnet, wenn die notwendigen Informationen zur Steuerung nur auf Anwendungsebene verfügbar sind oder der Algorithmus Anwendungsfunktionalität beinhaltet. Ein Beispiel dafür ist die Filterung von Anwendungsdaten anhand von Inhaltsinformationen. Generische Mechanismen sind dagegen unabhängig von Informationen einer bestimmten Anwendung bzw. können von mehreren Anwendungen gemeinsam genutzt werden. Beispiele dafür sind die verlustfreie Kompression von Anwendungsdaten und die Konvertierung von Bildformaten.

In ähnlicher Weise erfolgt eine Unterscheidung entsprechend der Platzierung der Mechanismen. So werden Filterung und Datenreduzierung in der Regel vor der Übertragung von Daten über eine schmalbandige Verbindung angewendet. Die Zwischenspeicherung von Daten in einem Cache zur Reduzierung der Antwortzeit erfolgt dagegen nach der Übertragung auf dem tragbaren Endgerät.

Adaption wird in diesem Klassifikationsschema also anhand der einzelnen Mechanismen zur Realisierung einer adaptiven Anwendung differenziert betrachtet. Für eine Vielzahl

von Mechanismen ist, zum Teil unter Berücksichtigung des konkreten Anwendungsfalls, eine eindeutige Zuordnung möglich. Mechanismen, die unabhängig von einem bestimmten Ort (z. B. Queuing) oder sowohl auf System- als auch auf Anwendungsebene (z. B. Filterung) einsetzbar sind, können aber auch in diesem Schema nicht eindeutig zugeordnet werden.

3.1.3 Klassifikation anhand einer Vier-Ebenen-Architektur

Die in [FDB+99] beschriebene Klassifikation erweitert das Schema aus Abschnitt 3.1.1. Das Kriterium der Klassifikation ist die Ebene der Systemarchitektur, auf welcher die Adaption stattfindet. Techniken werden in diesem Schema in vier Kategorien eingeordnet. Dies sind die Ebenen: Benutzer, Anwendung, Middleware und Kommunikation (Schichten 1 bis 4 nach OSI). Auf jeder der vier Ebenen wird eine Reihe von Techniken eingeordnet. Die Klassifikation erfolgt in nur einer Dimension, der Systemarchitektur. Damit wird die Sichtweise auf anwendungsbasierte und systembasierte Adaption ebenso wie auf anwendungsspezifische und generische Mechanismen verfeinert.

Entsprechend der Klassifikation kann der Benutzer abhängig von der Verbindung seine Aufgabe wählen, zwischen synchroner und asynchroner Zusammenarbeit wechseln oder eine Priorisierung paralleler Aktivitäten und Applikationen vornehmen. Auf der Anwendungsebene können komplexe oder ressourcenintensive Verarbeitungen verlagert werden (z. B. durch eine Vorverarbeitung von Daten im Festnetz oder der Filterung von Daten vor der Übertragung). Weiterhin kann zur Anpassung ein dynamisches Binden an neue Dienste erfolgen. Insbesondere können in Phasen der Abkopplung Stellvertreterdienste genutzt werden, die Daten aus einem Cache liefern. Weitere Anpassungsmöglichkeiten sind das Interaktionsmodell, die Struktur und die Anforderungen der Anwendung (z. B. durch Aushandlung neuer QoS-Anforderungen). Auf Middlewareebene können Techniken zum verzögerten Laden, Vorabladen, zur Filterung und Kompression und zum effizienten Einsatz von Protokollen eingesetzt werden. Auf der Ebene der Kommunikation wurden Techniken, wie die Auswahl von spezifischen Protokollen, zur Optimierung der Datenübertragung, zur Anpassung von Multicasts an die jeweilige Netzwerktechnologie, die Umordnung und Priorisierung von Daten und die Nutzung parallel verfügbarer Zugangsnetzwerke eingeordnet.

Das Klassifikationsschema besitzt eine heterogene Granularität hinsichtlich der Betrachtung von Adaptionsmechanismen. Zum einen werden einzelne Mechanismen identifiziert, zum anderen erfolgte eine grob-granulare Einordnung in weiter zerlegbare Kategorien (z. B. „Optimize for the characteristics of the network“ vs. „Prefetching into the cache“). Die Aufzählung der eingeordneten Techniken verdeutlicht, dass diesen zum Teil gleiche Mechanismen zugrunde liegen. So ist eine Priorisierung von Aktivitäten durch den Benutzer unter anderem auch durch eine Priorisierung der zu übertragenden Daten erreichbar. Außerdem gilt auch für diese Klassifikation die bereits angesprochene Problematik, dass Techniken bzw. Mechanismen nicht einer bestimmten Ebene zugeordnet werden können, sondern abhängig von Einsatz und Implementierung zu verschiedenen Ebenen gehören. Beispielsweise kann Caching auf Anwendungs-, Middleware- und Kommunikationsebene eingesetzt werden.

3.1.4 Gegenstand der Adaption

Die bisherigen Betrachtungen haben gezeigt, dass anhand der Kriterien „Ebene der Systemarchitektur“ und „Platzierung“ für viele Mechanismen eine eindeutige Klassifikation

nicht bzw. nur unter Betrachtung eines konkreten Anwendungsfalls möglich ist. Wie bereits diskutiert, ist die Platzierung von Mechanismen nicht statisch definierbar, sondern entscheidet im konkreten Anwendungsfall über die Effizienz des eingesetzten Mechanismus'. Die Platzierung ist somit selbst Gegenstand der Adaption.

Die Universalität eines Mechanismus' wird ebenfalls nicht durch den Mechanismus selbst bestimmt, sondern hängt im Allgemeinen von der Verfügbarkeit von Informationen für dessen Steuerung ab. Besteht beispielsweise die Möglichkeit der Konfiguration einer Filteroperation durch die Anwendung, so kann diese auch auf Systemebene eingesetzt werden. Ein Informationsaustausch zwischen System und Anwendung, z. B. durch einen Kontextdienst (siehe Abschnitt 2.4), ermöglicht einen universellen Einsatz der Mehrzahl der Mechanismen. Die Universalität eines Mechanismus' ist somit abhängig von der Implementierung des Gesamtsystems und eignet sich nur sehr bedingt zur Klassifikation von Mechanismen zur Adaption.

Nachfolgend wird ein alternatives Klassifikationsschema entwickelt, dem die Fragestellung zu Grunde liegt, welcher Bestandteil einer verteilten Anwendung durch einen Mechanismus verändert wird, um eine Adaption an die Ausführungsumgebung zu erreichen. Als Ausgangspunkt der Überlegungen dient die Sicht von Architekturbeschreibungssprachen auf verteilte Anwendungen (Abschnitt 2.5.4). Diese werden ganz allgemein als Menge von Komponenten betrachtet, zwischen denen Kommunikationsbeziehungen bestehen. Die Komponenten können auf unterschiedlichen Rechnern des verteilten Systems platziert sein und führen autonom Verarbeitungsschritte aus. Während der Verarbeitung kooperieren die verteilten Komponenten mit Hilfe von Interaktionen, um ein gemeinsames Ziel zu erreichen. Die Komponenten enthalten also sowohl Mechanismen zur lokalen Datenverarbeitung als auch Mechanismen zur Kooperation, d. h. zum Austausch von Daten. Die Funktionalität einer verteilten Anwendung wird somit durch die Verknüpfung von Mechanismen zur Datenverarbeitung und -übertragung erbracht.

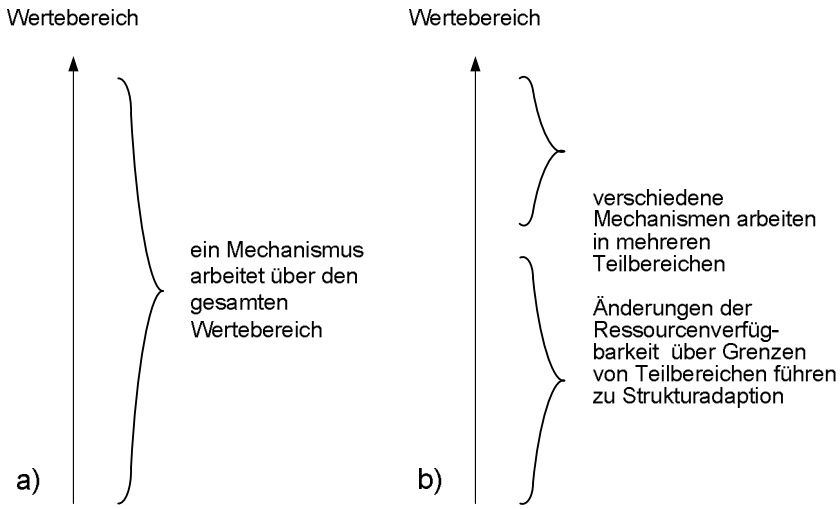


Abbildung 3-3: Wertebereiche von Mechanismen zur Adaption

Die Adaption verteilter Anwendungen kann also durch den gezielten Einsatz von Mechanismen zur Verarbeitung und Übertragung von Anwendungsdaten erreicht werden. Die einzelnen Mechanismen arbeiten in der Regel in einem bestimmten Wertebereich und bewirken bei entsprechender Parametrisierung ein adaptives Verhalten der verteilten Anwendung. Beispielsweise kann bei einem Mechanismus zur Konvertierung eines Bildes durch entsprechende Festlegung der Parameter für die x- und y-Koordinaten die Dateigrö-

ße des Bildes nach der Konvertierung gesteuert werden. Arbeitet ein Mechanismus über den gesamten geforderten Wertebereich, kann die Adaption allein durch Parametrisierung erreicht werden (siehe Abbildung 3-3a).

Im Allgemeinen liefern Mechanismen jedoch nur Ergebnisse in einem kleineren Wertebereich, als dies für die Adaption gefordert wird. Beispielsweise existiert eine untere Grenze zur Reduzierung der Dateigröße eines Bildes. Unterhalb dieser Grenze ist der Bildinhalt nicht mehr erkennbar, d. h. die Konvertierung liefert keine sinnvollen Ergebnisse mehr. In diesem Fall können mehrere Mechanismen derart kombiniert werden, dass sich ihre Wertebereiche ergänzen und damit den Wertebereich der Adaption vergrößern (siehe Abbildung 3-3b). Beispielsweise kann durch den Mechanismus „Ersetzen“ ein Bild durch eine alternative Repräsentation (z. B. ein Rahmen oder ein Text zur Beschreibung des Bildinhaltes) ersetzt werden, um die Dateigröße des Bildes weiter zu reduzieren. Der Wertebereich bezieht sich im obigen Beispiel auf die Größe des adaptierten Bildes. Ebenso könnte abhängig von den Anforderungen, die durch die Anwendung an die Adaption gestellt werden, der Wertebereich für die Qualität des Bildes, den Ressourcenverbrauch oder die Verarbeitungszeit einer Adaptionsoperation betrachtet werden.

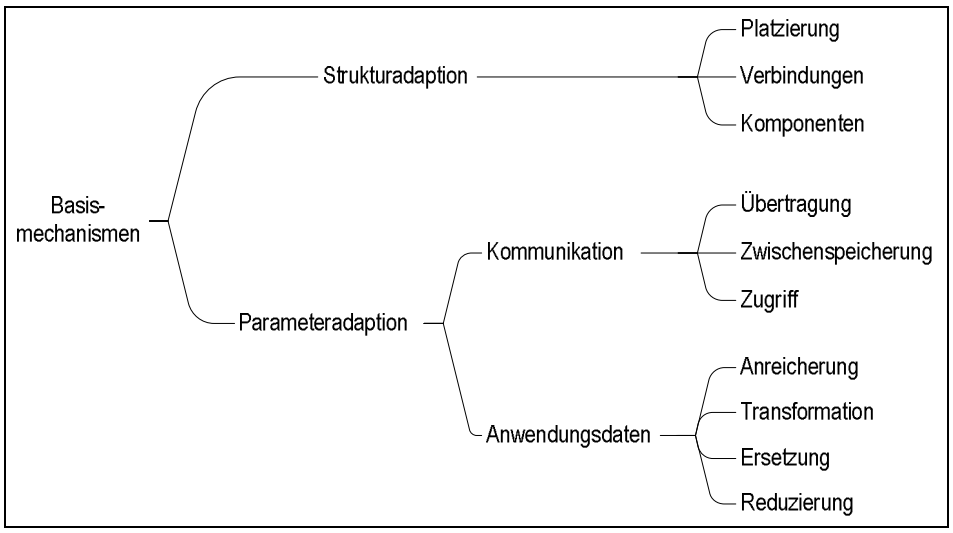


Abbildung 3-4: Mechanismen zur Adaption

Entsprechend dieser Überlegungen kann Adaption zum einen durch die *Parametrisierung* von Mechanismen zur Verarbeitung und Übertragung von Anwendungsdaten, zum anderen durch *Strukturadaption*, d. h. die Änderung der Konfiguration der verteilten Anwendung erreicht werden. Somit können drei Hauptklassen von Gegenständen der Adaption identifiziert werden. Diese sind 1. die Anwendungsdaten, 2. deren Übertragung, d. h. die Kommunikation zwischen Komponenten einer verteilten Anwendung sowie 3. die Anwendungsstruktur.

Mechanismen der Strukturadaption sind das Hinzufügen, Entfernen, Umordnen und Replizieren[6] von Komponenten und die Auswahl und dynamische Änderung der Platzierung einzelner Komponenten. Damit verbunden ist die Anpassung von Verbindungen zwischen Komponenten. Parameteradaption umfasst Mechanismen zur Adaption von Anwendungsdaten und zur Kommunikation zwischen Komponenten bzw. Mechanismen. Anwendungs-

[6] Umordnen und Replizieren können als Kombination von Hinzufügen und Entfernen realisiert werden, stellen aber wesentliche Adaptionsmechanismen dar und werden deshalb explizit aufgeführt.

daten können verworfen, ersetzt, transformiert und angereichert werden. Zur Adaption der Kommunikation zählen Mechanismen zur Übertragung, Zwischenspeicherung und zur Verlagerung des Zugriffszeitpunktes. Abbildung 3-4 enthält ein Schema zur Klassifizierung der Basismechanismen zur Adaption anhand des Gegenstandes der Adaption.

3.2 Klassifikation von Basismechanismen zur Adaption

In den folgenden Abschnitten werden Basismechanismen zur Adaption identifiziert, diskutiert und in das Klassifikationsschema eingeordnet. Die Basismechanismen stellen dabei Klassen von Mechanismen dar, für die verschiedene Implementierungen existieren können. Die Benennung der Basismechanismen erfolgt anhand der in der Literatur verwendeten Begriffe. Zum Teil erfordert das Klassifikationsschema jedoch eine weitere Präzisierung bzw. genauere Unterscheidung der Mechanismen als dies durch die Semantik der Begriffe vorgegeben wird. Deshalb werden die Basismechanismen nachfolgend für die weiteren Betrachtungen im Rahmen dieser Arbeit definiert. Die Einordnung der Mechanismen hängt außerdem teilweise von der Sichtweise auf die Mechanismen ab. Getroffene Entscheidungen werden deshalb in den entsprechenden Abschnitten diskutiert. Ausgewählte Beispiele zur Realisierung der Mechanismen aus den in Kapitel 2 beschriebenen Forschungsprojekten dienen der Veranschaulichung der Klassifikation.

3.2.1 Adaption von Anwendungsdaten

Die Möglichkeiten zur Adaption von Anwendungsdaten sind sehr vielfältig und werden unter anderem durch die Zusammensetzung der Daten bestimmt. Im einfachsten Fall sind die Daten atomar, d. h. diese besitzen keine Struktur und können nicht sinnvoll zerlegt werden. Die Eigenschaft ist anwendungsabhängig. Beispielsweise kann ein Bild als atomar betrachtet werden, da es zu einem Zeitpunkt vollständig am Bildschirm dargestellt werden kann. Es besitzt also keine zeitliche Struktur. Implizit besitzt ein Bild jedoch immer eine räumliche Struktur und kann z. B. in Segmente (z. B. ein Stadtplan) oder Farbebenen zerlegt werden. Außerdem können zusammengehörende Datenobjekte als atomar gekennzeichnet werden, wenn sie logisch untrennbar zusammengehören (z. B. ein Text-Eingabeelement und ein beschreibender Text innerhalb einer Web-Seite).

Nicht-atomare Datenobjekte sind zusammengesetzt und besitzen somit eine Struktur. Die Zusammensetzung kann in enger Kopplung, d. h. innerhalb eines Datensatzes bzw. einer Datei oder in loser Kopplung erfolgen. Werden die Datenobjekte eng gekoppelt, können die Strukturinformationen entweder implizit (z. B. Sätze und Abschnitte innerhalb eines Textes) oder explizit durch eine Strukturbeschreibung vorliegen (z. B. in Form einer Formatbeschreibung). Durch eine explizite Strukturbeschreibung können beliebige Strukturen insbesondere auch hierarchische Strukturen definiert werden (z. B. in einer E-Mail). Erfolgt eine Zusammensetzung von Datenobjekten in loser Kopplung, befinden sich alle Datenobjekte in separaten Dateien. Die Zusammensetzung kann entweder in Form von Verweisen innerhalb der Datenobjekte (z. B. in Web-Seiten) oder durch eine explizite Strukturbeschreibung, d. h. getrennt von den beschriebenen Daten erfolgen (siehe Abbildung 3-5).

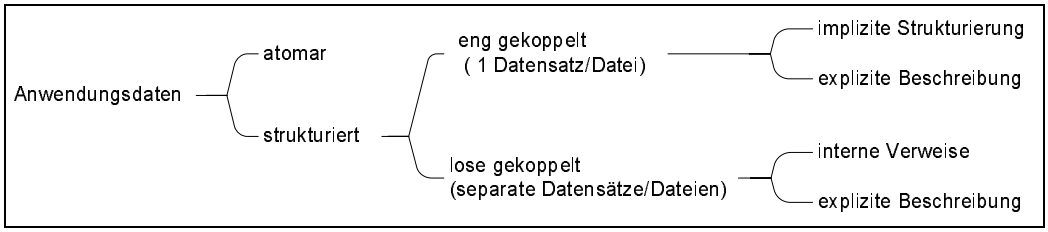


Abbildung 3-5: Möglichkeiten der Zusammensetzung von Anwendungsdaten

Weitere Merkmale von Datenobjekten sind der Datentyp (z. B. Text, Bild, Audio, Video), das Datenformat (z. B. GIF oder JPEG für Bilddaten) sowie die Daten selbst und deren Kodierung. Eine Einordnung der Adaptionsmechanismen entsprechend der Merkmale Struktur, Typ, Format und Inhalt führt aber für einige der Mechanismen zu keiner eindeutigen Zuordnung. Beispielsweise kann eine Filterung von Daten auf Basis aller dieser Merkmale erfolgen. Es müssen also alternative Kriterien definiert werden. Dazu dient die Überlegung, dass Anwendungsdaten in einer bestimmten Qualität und mit einem bestimmten Informationsgehalt vorliegen. Eine Adaption ist also durch das Erhöhen, das Beibehalten oder das Reduzieren des Informationsgehaltes der Daten möglich. Reduziert werden können Daten durch das Verwerfen eines Teils der enthaltenen Informationen bzw. durch das Ersetzten durch alternative Informationen. Je nach Mechanismus wird aber eine Bewahrung eines möglichst großen Teils der enthaltenen Informationen angestrebt. Unter Beibehaltung des Informationsgehaltes kann eine Adaption durch die Transformation der Daten erfolgen. Eine vierte Klasse von Mechanismen zur Adaption umfasst die Anreicherung der Originaldaten durch zusätzliche (Meta-) Informationen. Nachfolgend werden Mechanismen zur Adaption der Anwendungsdaten entsprechend der Klassen Reduzierung, Transformation, Ersetzung und Anreicherung diskutiert (siehe Abbildung 3-4).

3.2.1.1 Reduzierung

Mechanismen der nachfolgenden Klassen resultieren in einer Reduzierung des Informationsgehaltes von Datenobjekten. Neben dem primären Ziel der Reduzierung der Datenmenge können die Mechanismen auch zur Anpassung an Eigenschaften des Endgerätes, Benutzereinstellungen und die Anwendungssituation verwendet werden. Die Unterklassen enthält Abbildung 3-6.

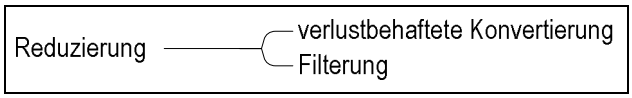


Abbildung 3-6: Mechanismen zum Verwerfen von Anwendungsdaten

Filterung

Unter Filterung wird die Auswahl von Daten oder Teildaten und das Verwerfen der ausgewählten Daten verstanden. Entsprechend ihrer Zusammensetzung können entweder ganze Datenobjekte oder Teile einer Datenstruktur gefiltert werden. Eine Filterung kann auf Basis von Informationen über die Struktur, den Typ, das Format, die Semantik der Daten sowie externer Informationen erfolgen. Bei strukturierten Daten kann beispielsweise die Position eines Datenobjektes innerhalb der Hierarchie als Information über die Relevanz des Datenobjektes interpretiert und für eine Filterung genutzt werden [Wat95]. Ein weiteres Beispiel dafür ist das Verwerfen von P- und B-Frames eines MPEG-Stromes [Zen99]. Eine Filterung auf Basis des Datentyps dient unter anderem zur Anpassung an die

unterstützten multimedialen Formate eines mobilen Endgerätes. So können z. B. Audio- und Videodaten aus einer Web-Seite oder E-Mail gefiltert werden, wenn diese Datentypen nicht vom Endgerät repräsentiert werden können. Ähnliches gilt für die Ebene des Datenformates. Die Semantik der Anwendungsdaten liefert die Informationen für eine sehr differenzierte Filterung. So können Textdokumente anhand von Schlagwörtern und E-Mail-Nachrichten entsprechend ihres Themas oder des Senders gefiltert werden. Außerdem ist eine Filterung auf Basis externer Informationen möglich. Beispielsweise können Datenpakete verworfen werden, wenn die Länge der Warteschlangen eines Routers bestimmte Schwellwerte überschreitet, was als Indikator für die Entstehung eines Staus interpretiert wird [FlJ93].

Verlustbehaftete Konvertierung

Basismechanismen zur verlustbehafteten Konvertierung führen auf Datenobjekten datenabhängige Operationen aus, die in einer Reduzierung des Informationsgehaltes resultieren. Im Unterschied zur Filterung werden Daten nicht ausgewählt und verworfen. Konvertierungsmechanismen beziehen in der Regel die Originaldaten in ihre Operationen ein und verknüpfen diese zu einem verwandten Datenobjekt des gleichen Typs. Die Möglichkeiten für Konvertierungsmechanismen sind vielfältig und hängen vom Typ der Daten ab. Bilder können beispielsweise skaliert oder in der Farbtiefe reduziert werden [FGC+98]. Ähnliche Mechanismen können auf Videodaten angewendet werden. Für diesen Datentyp kann zusätzlich die Framerate angepasst werden. Audiodaten können in ihrer Samplefrequenz und Amplitude konvertiert werden.

Mechanismen zur Umwandlung von Datenformaten werden zum Teil als Formatkonvertierung bezeichnet. Beispiele dafür sind die Umwandlung eines JPEG-Bildes in ein BMP-Bild oder die Umwandlung von Audiodaten vom MP3- in das WAV-Format. In den genannten Beispielen wird nur das Datenformat verändert, der Informationsgehalt bleibt dagegen konstant. Diese Mechanismen werden deshalb in der Klassifikation von einer verlustbehafteten Konvertierung unterschieden und können in die Klasse Formattransformation eingeordnet werden (siehe Abschnitt 3.2.1.3).

Mechanismen zur verlustbehafteten Kompression von Daten, abhängig vom Datentyp (z. B. JPEG), werden intuitiv in diese Klasse eingeordnet. Das Klassifikationsschema erlaubt jedoch eine weitere Zerlegung dieser Mechanismen, die in unterschiedliche Klassen zur Adaption der Anwendungsdaten eingeordnet werden können. In der Regel wird eine verlustfreie Kompression mit der Transformation und Filterung von Daten kombiniert. Beispielsweise werden beim JPEG-Verfahren für Bilder die Daten zunächst entsprechend umkodiert, dann vom Zeit- in den Frequenzraum transformiert und danach gefiltert. Auf die gefilterten Daten wird anschließend ein verlustfreies Komprimierungsverfahren angewendet [Ste99] (siehe auch Abschnitt 3.2.1.3).

3.2.1.2 Ersetzung

Mechanismen zur Ersetzung eines Datenobjektes tauschen die Originaldaten gegen alternative Daten aus. Die alternativen Daten können als Synonym der originalen Daten betrachtet werden. Dabei wird die Bewahrung eines möglichst großen Teils des Informationsgehaltes der Originaldaten angestrebt. Die alternativen Daten können dabei explizit zur Verfügung stehen, müssen aus implizit verfügbaren Daten gewonnen oder aus den Originaldaten generiert werden. Entsprechend Abbildung 3-7 können die folgenden drei Klassen von Mechanismen unterschieden werden: Generierung, Extrahierung und Auswahl.

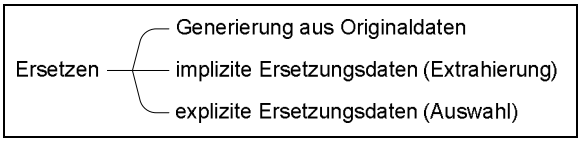


Abbildung 3-7: Möglichkeiten der Verfügbarkeit von Ersetzungsdaten

Generierung

Mechanismen dieser Klasse generieren aus den Informationen des originalen Datenobjektes alternative Daten. Im Gegensatz zur verlustbehafteten Konvertierung besitzt das erzeugte Datenobjekt meist einen anderen Typ als die Originaldaten. So kann aus Audiodaten eine Textrepräsentation erzeugt werden, die die gesprochenen Worte in der Audiodatei beinhalten (Speech-to-Text) [KiF00]. Ein weiteres Beispiel ist die Erzeugung einer Vektorgrafik aus einem Bild mit Hilfe von Konturerkennungsverfahren.

Extrahierung

Durch die Extrahierung werden implizit in den Originaldaten enthaltene Informationen zur Ersetzung verwendet. So kann ein Datenobjekt durch seinen Dateinamen oder ein Bild durch einen Rahmen mit den Maßen des Bildes ersetzt werden. Dazu muss auf die im Bild enthaltenen Informationen zu Länge und Breite des Bildes zugegriffen werden (siehe z. B. Futures und Outlines in [Wat95], Ersetzungtext für E-Mail-Anhänge in [SpS00]).

Auswahl

In der Klasse Auswahl werden Mechanismen eingeordnet, die ein Datenobjekt durch eine explizit verfügbare alternative Repräsentation ersetzen. Für ein Datenobjekt müssen dazu mehrere Datenobjekte verfügbar sein, die alternative Repräsentationen eines Inhaltes darstellen. Ein Beispiel ist die Ersetzung eines Bildes durch einen beschreibenden Text wie dies unter anderem in HTML durch das ALT-Element innerhalb des IMG-Elementes unterstützt wird (siehe [FoB96]). Eine notwendige Voraussetzung für die Ersetzung ist die Verfügbarkeit der alternativen Daten.

3.2.1.3 Transformation

Mechanismen zur Transformation verändern die interne Repräsentation von Datenobjekten, verwerfen aber keine Originaldaten. Transformiert werden können die Struktur, das Format sowie die Kodierung. Zu den einzelnen Klassen von Transformationsmechanismen können noch weitere Unterklassen definiert werden (siehe Abbildung 3-8). Diese werden nachfolgend beschrieben. Alle Transformationen besitzen die Gemeinsamkeit, dass es einen inversen Mechanismus gibt, der die Originaldaten wieder herstellen kann.

Strukturtransformation

Mechanismen zur Strukturtransformation überführen die Struktur zusammengesetzter Anwendungsdaten zwischen unterschiedlichen Darstellungsformen (siehe Abbildung 3-5). Dabei wird nur die Verknüpfung zwischen den Datenobjekten verändert, die Datenobjekte selbst werden nicht verarbeitet. Strukturtransformationen können eingesetzt werden, um eine effizientere Verarbeitung zu erreichen, nicht unterstützte Datenstrukturen in unterstützte Strukturen zu überführen oder durch eine Restrukturierung von Datenobjekten eine

angepasste Übertragung und Darstellung zu ermöglichen. Es können drei grundsätzliche Mechanismen unterschieden werden (siehe Abbildung 3-8).

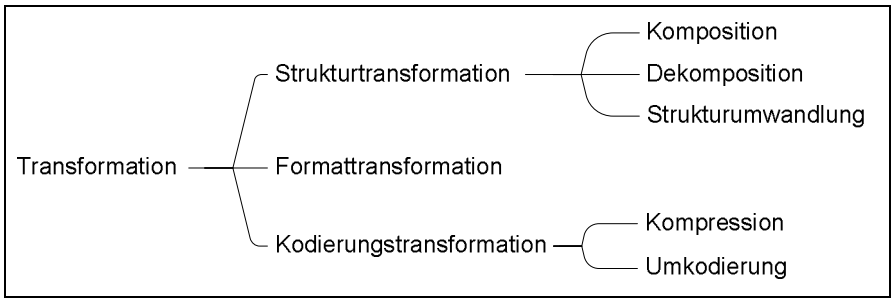


Abbildung 3-8: Mechanismen zur Transformation von Datenobjekten

Strukturumwandlung[7]

Das Ergebnis der Strukturumwandlung ist ein Datenobjekt in einer von der Ausgangsstruktur verschiedenen Darstellung. Ein Beispiel für die Transformation von Datenobjekten ist die Überführung eines XML-Dokumentes in eine Baumstruktur entsprechend des Document Object Models (DOM) [W3C04]. Ein nur implizit strukturierter Text kann in eine explizite Struktur überführt werden. Auf Basis der expliziten Struktur kann dann z.B. ein Inhaltsverzeichnis aufgebaut und für Kapitel, Abschnitte usw. eine effiziente Präsentation und Navigation durch den Text erreicht werden (z. B. für Web-Dokumente [BGP01]).

Dekomposition

Durch die Dekomposition strukturierter Daten werden diese in separate (evtl. atomare) Datenobjekte zerlegt, die unabhängig und durch unterschiedliche Mechanismen verarbeitet bzw. adaptiert werden können. Das Resultat der Dekomposition ist demnach eine Sequenz von Datenobjekten. Beispielsweise kann eine E-Mail-Nachricht in einen Nachrichtenkopf, den Nachrichtentext sowie die Anhänge zerlegt werden. Weiterhin kann ein zusammengefasster Datenstrom von Video- und Audiodaten in die einzelnen Bestandteile zerlegt werden, um diese getrennt zu verarbeiten und anzupassen.

Komposition

Separate Datenobjekte können durch Komposition zu einer Datenstruktur zusammengefasst werden, um diese beispielsweise als Ganzes zu verarbeiten bzw. anzupassen. Insbesondere wenn zu einem früheren Zeitpunkt eine Dekomposition erfolgte, kann das strukturierte Datenobjekt durch Komposition wieder hergestellt werden. So kann eine E-Mail-Nachricht zunächst zerlegt werden. Nach einer separaten Adaption und Übertragung der Teildaten kann z. B. auf dem Endgerät aus den Teildaten wieder eine E-Mail-Nachricht erzeugt werden [SpS00]. Ein weiteres Beispiel für die Komposition ist die Zusammenfassung separater Audio- und Videoströme zu einem integrierten Strom.

[7] Eine Strukturumwandlung kann durch die Mechanismen Dekomposition und Komposition erreicht werden, soll hier aber aufgrund der eigenständigen Semantik als eigenständige Klasse von Mechanismen betrachtet werden.

Formattransformation

Mit dem Begriff Formattransformation werden Mechanismen zur Überführung von Datenobjekten eines Formates in ein anderes Format beschrieben. Der Typ der Daten wird durch die Formattransformation nicht verändert. Eine Änderung des Formates dient zur Anpassung bei inkompatiblen Verarbeitungsoperationen und bei fehlender Unterstützung bestimmter Formate durch mobile Endgeräte. Beispielsweise können WAP-Geräte nur Bilder im WBMP-Format darstellen, wodurch eine Umwandlung anderer Bildformate in dieses Format notwendig wird. Ein weiteres Beispiel ist die Umwandlung von Postscript Dokumenten in RTF oder HTML Dokumente, wie dies in [FGC+98] beschrieben wird. Diese Formattransformation stellt einen Grenzfall in Bezug auf die verlustbehaftete Konvertierung dar, da bei der Transformation eines ausdrucksstärkeren Formates in ein weniger ausdrucksstarkes Format Informationen verloren gehen. Da jedoch das primäre Ziel die Umwandlung in ein alternatives Format und nicht die Reduzierung des Informationsgehaltes ist, werden auch diese Mechanismen in die Klasse Formattransformation eingeordnet.

Kodierungstransformation

Mechanismen zur Kodierungstransformation verändern die interne Repräsentation von Anwendungsdaten, ohne den Informationsgehalt zu reduzieren. Im Gegensatz zur Formattransformation wird das Format der Anwendungsdaten nicht verändert. Insbesondere kann ein Datenformat verschiedene Kodierungen unterstützen. Es können zwei Klassen von Mechanismen unterschieden werden (siehe Abbildung 3-8). Zum einen können Daten mit dem Ziel der Verringerung des Datenvolumens komprimiert werden. Die Kompression ist verlustfrei, d. h. durch eine inverse Operation können die Ausgangsdaten wieder hergestellt werden. Zum anderen können Anwendungsdaten umkodiert werden, um Inkompatibilitäten zwischen Verarbeitungsoperationen auszugleichen oder Daten in eine vom Endgerät unterstützte Kodierung zu überführen.

Kompression

Wie bereits beschrieben, wird durch die Kompression eine verlustfreie Reduzierung des Volumens von Anwendungsdaten ermöglicht. Diese Verfahren werden auch als Entropiekodierung bezeichnet [Ste99]. Die Dekompression ist die inverse Operation der Kompression und stellt die Originaldaten wieder her. Kompressionsverfahren werden oft in Kombination mit verlustbehafteten Kompressionsverfahren zu hybriden Verfahren kombiniert (siehe [Ste99]). So werden im JPEG-Verfahren [Wal91] die Daten nach der Quantisierung durch ein verlustfreies Verfahren komprimiert. Beispiele für verlustfreie Kompressionsverfahren sind die Huffmann- [Huf52] und die Lauflängenkodierung [Gol66].

Umkodierung

Im Gegensatz zur Kompression ist das Ziel einer Umkodierung die Veränderung der internen Repräsentation von Daten. Dadurch werden Inkompatibilitäten ausgeglichen oder eine effizientere Verarbeitung der Daten ermöglicht. Beispiele sind eine Transformation von Bilddaten zwischen unterschiedlichen Farbräumen oder vom Zeit- in den Frequenzbereich (z. B. mit Hilfe einer diskreten Kosinustransformation [Ste99]) sowie die Umkodierung von ASCII-Text in Unicode.

3.2.1.4 Anreicherung

Der Begriff Anreicherung umfasst Mechanismen, die den Informationsgehalt eines Datenobjektes durch zusätzliche Informationen erhöhen. Aus der Perspektive eines gegebenen Datenobjektes kann dies durch Mechanismen zur Aggregation und Annotation erfolgen (siehe Abbildung 3-9).

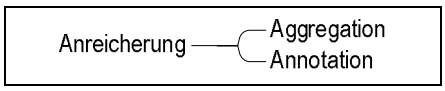


Abbildung 3-9: Mechanismen zur Anreicherung von Anwendungsdaten mit Informationen

Aggregation

Durch Mechanismen zur Aggregation wird ein Datenobjekt mit Informationen aus weiteren separaten Datenobjekten angereichert. Dabei werden bestimmte Informationen jedes Datenobjektes ausgewählt, und in das Zieldatenobjekt integriert. Das Zieldatenobjekt kann bespielsweise ein Template sein, welches Platzhalter für die zu integrierenden Informationen enthält. Ein Beispiel ist die Aggregation von Ergebnissen mehrerer Suchmaschinen zu einer Seite mit einer Zusammenfassung der jeweils besten Suchergebnisse oder die Zusammenfassung von Web-Seiten mit ähnlichen Inhalten wie z. B. Veranstaltungsseiten einer Region [FGC+97].

Annotation

Der Begriff Annotation beschreibt Mechanismen, die ein Datenobjekt mit zusätzlichen Informationen anreichern. Beispielsweise wird in [GrF97] ein Mechanismus beschrieben, der reguläre Ausdrücke verwendet, um in HTML-basierten Web-Seiten passende Kombinationen von Buchstaben hervorzuheben. Sowohl die regulären Ausdrücke als auch die Art des Hervorhebens (z. B. durch Rotfärbung des Textes) werden vom Benutzer definiert. Ein weiteres Beispiel ist die Markierung von Telefonnummern, Adressen u. ä. in Web-Seiten durch die Anwendung von Heuristiken [STH+01].

3.2.2 Adaption der Kommunikation

Mechanismen zur Adaption der Kommunikation arbeiten zum einen auf der Ebene der Übertragungsprotokolle. Zum anderen wird die Verfügbarkeit von Datenobjekten und Zwischenergebnissen durch Mechanismen zur Zwischenspeicherung und zur Steuerung des Zugriffs auf Daten angepasst.

3.2.2.1 Übertragung

Auf den unteren Schichten von Protokollen (Schichten 1 bis 4 nach OSI) werden grundlegende Dienste zur Datenübertragung realisiert. Dies sind z. B. Mechanismen zur Aufteilung von Daten in Pakete und Rahmen, zur Fehlererkennung und –behandlung, zur Regulierung des Informationsflusses zwischen Netzknoten bzw. Sender und Empfänger und zur Staubbehandlung. Protokolle wurden zum Teil für bestimmte Übertragungsmedien konzipiert und treffen bestimmte Annahmen, die aber nicht den Eigenschaften aller Übertragungsmedien entsprechen. Ein Beispiel ist die Staubehandlung in TCP. Außerdem unterstützen konventionelle Protokolle nicht die besonderen Eigenschaften drahtloser oder asymmetrischer Kommunikationsmedien (siehe Abschnitt 2.1.2). Insbesondere auf der Sicherungs- und Transportschicht können Protokollparameter und Mechanismen zur

Fehlerbehandlung in Abhängigkeit des Übertragungsmediums angepasst werden. Zumeist sind in dieser Klasse angepasste Protokolle einzuordnen, die Mechanismen mehrerer Unterklassen kombiniert einsetzen. Eine Übersicht der Unterklassen enthält Abbildung 3-10.

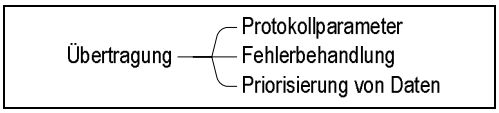


Abbildung 3-10: Mechanismen zur Adaption der Übertragung von Anwendungsdaten

Protokollparameter

Diese Klasse beinhaltet Mechanismen, die Parameter von Kommunikationsprotokollen an das aktuelle Übertragungsmedium anpassen. Parameter sind neben weiteren die Rahmen- und Paketgröße, Fenstergrößen zur Flusssteuerung, Zeitlimits (Timeouts) und die Größe von Werten in Paketköpfen. So werden in [YaB94] Pakete auf der Basisstation zerlegt bzw. zusammengefasst, um eine optimale Paketgröße für die Übertragung sowohl zum mobilen Endgerät, als auch zu Knoten im Festnetz zu erzeugen. In [SuB01] werden entsprechend der verfügbaren Netzwerkverbindung Timeout-Zeiten und die Größe des Initial Windows des TCP-Protokolls angepasst. Für UDP sieht die Lösung eine optionale Berechnung der Checksumme sowie einen Schlafzustand vor, in dem Pakete zwischenspeichert werden, bis eine Verbindungsunterbrechung beendet bzw. eine Verbindung mit den erforderlichen Eigenschaften hergestellt wird. Buffering ist ein Mechanismus zur Sammlung und Zusammenfassung mehrerer kleiner Pakete zu einem größeren Paket, um eine bessere Ausnutzung der verfügbaren Bandbreite zu erreichen [FoZ94]. In Mobile-TCP [Haa97] und [WaT98] werden zwischen mobilen Endgeräten und der Basisstation vereinfachte Protokollköpfe eingesetzt, um den Protokollaufwand zu reduzieren.

Fehlerbehandlung

Mechanismen zur Fehlerbehandlung werden in Protokollen eingesetzt, um in angepasster Weise Übertragungsfehler zu behandeln. Generell können hier fehlerkorrigierende und fehlerbehandelnde Verfahren unterschieden werden. In Mobile-TCP [Haa97] wird ein vereinfachtes fehlerbehandlendes Verfahren eingesetzt. Dabei wird angenommen, dass die letzte Verbindung zum mobilen Endgerät eine direkte Verbindung zur Basisstation ist. Das AIRMAIL-Protokoll [APL+95] setzt auf der Sicherungsschicht eine Kombination von fehlerkorrigierenden und fehlerbehandelnden Verfahren ein. Je nach Fehlerhäufigkeit und Art können verschiedene Stufen der Kombination eingesetzt werden, um ein optimiertes Verhältnis zwischen Redundanz und Übertragungswiederholungen zu erreichen. Eine direkte Erweiterung der Fehlerbehandlungsmechanismen in TCP wurde durch erweitere Empfangsbestätigungen [MMF+96, KeM96], explizite Stauinformationen [RFB01] und durch explizite Verlustinformationen erreicht [BaK98, DiJ01a], die ebenfalls in den Bestätigungen kodiert wurden.

Priorisierung von Datenströmen

Die Priorisierung von Daten bezeichnet Mechanismen, die Datenströme in verschiedene Klassen einordnen und mit unterschiedlicher Dienstgüte übertragen. Dies kann erreicht werden, indem Daten höherer Priorität gegenüber Daten geringerer Priorität bevorzugt übertragen werden. In Rover, dem Trickle Reintegration Mechanismus und in [HuH95] werden verschiedene Dienstklassen für Daten im Rahmen abgekoppelter Operationen

betrachtet (siehe Abschnitt 2.1.1.2). An Daten werden dabei entsprechend ihrer Bedeutung für ein Weiterarbeiten auf dem mobilen Rechner verschiedene Prioritäten vergeben.

3.2.2.2 Zwischenspeicherung

Mechanismen zur Zwischenspeicherung von Anwendungsdaten erhöhen zum einen die lokale Verfügbarkeit von Datenobjekten und Zwischenergebnissen. Zum anderen können Daten über einen begrenzten Zeitraum zwischengespeichert werden, um z. B. Verbindungsunterbrechungen zu behandeln. Es können die drei Klassen Queueing, Caching und Protokollierung unterschieden werden (siehe Abbildung 3-11).

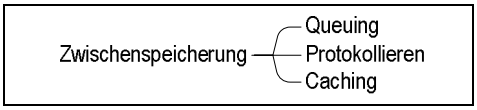


Abbildung 3-11: Mechanismen zur Zwischenspeicherung von Anwendungsdaten

Caching

Unter Caching werden Mechanismen eingeordnet, die Datenobjekte in einem Cache begrenzter Größe temporär zwischenspeichern. Auf die gespeicherten Datenobjekte kann über einen Schlüssel wahlfrei zugegriffen werden. Cachingmechanismen verwenden unterschiedliche Verfahren, um bei zu wenig Speicherplatz enthaltene Anwendendungsobjekte zum Löschen auszuwählen. Darunter zählen Least Recently Used (LRU) und Least Frequently Used (LFU). Für Caching können räumliche und zeitliche Abhängigkeiten in Datenzugriffen ausgenutzt werden. Zeitliche Abhängigkeiten sind gegeben, wenn auf ein gleiches Datenobjekt in kurzen Zeitabständen mehrmals zugegriffen wird. Räumliche Abhängigkeiten bestehen zum Beispiel bei verlinkten Dokumenten. Unter der Annahme, dass einer der Verweise als nächstes verfolgt wird, können beispielsweise Web-Seiten bis zu einer bestimmten Verweistiefe in den Cache geladen werden [Wat95].

Caching kann genutzt werden, um die zu übertragende Datenmenge und damit gleichzeitig die Antwortzeit zu reduzieren, indem eine mehrfache Übertragung über ein schmalbandiges Netz durch lokale Zwischenspeicherung vermieden wird (z. B. als Browsercache). Weiterhin kann ein Cache eingesetzt werden, wenn aufwendige Verarbeitungsoperationen auf ein Datenobjekt angewendet werden, auf das mehrmals zugegriffen wird. Mechanismen dieser Klasse werden in vielen Projekten vor allem in Kombination mit anderen Mechanismen genutzt. Im Snoop-Protokoll wird Caching benutzt, um nicht bestätigte Pakete auf der Basisstation zwischenzuspeichern und bei Paketverlusten eine Übertragungswiederholung durchzuführen, bevor beim Sender im Festnetz eine Staubehandlung ausgelöst wird [BPS+97a]. Im verteilten Dateisystem CODA wird Caching in Kombination mit Vorabladen eingesetzt, um bei Abkopplungen die lokale Verfügbarkeit von Dateien zu sichern [Kis96].

Queuing

Queuing-Mechanismen ermöglichen das Einstellen und Entnehmen von Datenobjekten in bzw. aus einer Warteschlange. In dieser Form der Zwischenspeicherung erfolgt im Gegensatz zu Caching eine Ordnung der zwischengespeicherten Daten (z. B. entsprechend der Reihenfolge des Eintreffens oder anhand von Prioritäten). Der Zugriff erfolgt nicht wahlfrei, sondern entsprechend der Ordnung auf das nächste Element. Queuing kann unter

anderem eingesetzt werden, um Nachrichten oder Datenobjekte während der Übertragung über unzuverlässigen Verbindungen zwischenzuspeichern [JTK97, SKS+99].

Protokollieren

Mechanismen zur Protokollierung sind eng verwandt mit Queuing-Mechanismen und können insbesondere durch Warteschlangen realisiert werden. Eine Unterscheidung der beiden Klassen Queuing und Protokollieren erfolgt aufgrund der unterschiedlichen Anwendungsbereiche der Mechanismen. Queuing wird überwiegend zur temporären Zwischenspeicherung von Datenobjekten (z. B. RPC-Nachrichten) zur Überbrückung von Verbindungsunterbrechungen verwendet und zielt damit auf eine robuste Übertragung der Daten. Protokollierung unterstützt dagegen ein autonomes Arbeiten, indem lokal ausgeführte Verarbeitungsschritte protokolliert und nach einem Verbindungsaufbau an einen Server im Festnetz vermittelt werden (siehe Abschnitt 2.1.1.2).

3.2.2.3 Verlagerung des Zugriffs

Mechanismen zur Steuerung des Zugriffs entkoppeln die Übertragung von Datenobjekten von der Verarbeitung dieser Daten. Generell kann die Übertragung vor oder nach der Verarbeitung der Daten stattfinden. Dadurch erfolgt eine Verteilung der Zugriffe entsprechend der verfügbaren Bandbreite. Die Mechanismen erreichen in der Regel zwar keine Reduzierung der Netzlast, ermöglichen aber eine bessere bzw. effizientere Ausnutzung dieser und erreichen eine Verbesserung der Antwortzeiten und damit der subjektiv wahrgenommenen Leistungsfähigkeit von Verbindungen [FoZ94]. Die Klassen von Mechanismen zur Verlagerung des Zugriffs auf Datenobjekte werden in Abbildung 3-12 dargestellt.

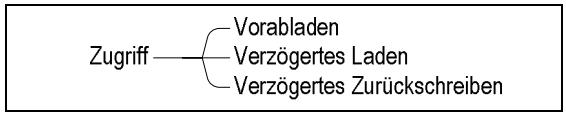


Abbildung 3-12: Mechanismen zur Verlagerung des Zugriffs auf Anwendungsdaten

Vorabladen

Mechanismen zum Vorabladen verlagern den Zeitpunkt der Anforderung und Übertragung vor den Zeitpunkt der Nutzung von Datenobjekten. Im Allgemeinen müssen dazu Informationen über zukünftige Zugriffe auf Daten vorliegen. Diese können entweder durch den Benutzer geliefert werden oder müssen durch Beobachtung des Benutzers bzw. unter Berücksichtigung anwendungsabhängiger Informationen gewonnen werden. In CODA wird ein sogenannter Hoarding-Mechanismus eingesetzt, um Dateien für ein abgekoppeltes Arbeiten auf dem Endgerät verfügbar zu machen. Die Informationen über zukünftig benötigte Dateien liefert bei diesem Ansatz der Nutzer in Form einer Konfigurationsdatei [Kis96]. In webbasierten Systemen kann ein Vorabladen anhand der Link-Struktur erfolgen, ohne den Benutzer explizit einzubeziehen. Entsprechend der Verweise können Dateien in bis zu einer festgelegten Hierarchie-Tiefe vorab geladen werden [Wat94b]. Vorabladen verbessert die Antwortzeit für Anfragen, falls die Voraussagen zutreffen und erhöht die Verfügbarkeit von Daten bei Abkopplungen. Für das Laden werden aber zusätzliche Ressourcen (z. B. zur Verarbeitung oder zur Übertragung) benötigt, die verschwendet werden, wenn die vorab geladenen Daten nie angefordert bzw. verarbeitet werden.

Verzögertes Laden

Durch Mechanismen zum verzögerten Laden wird der Zeitpunkt des Ladens von Datenobjekten hinter den Zeitpunkt von deren Nutzung verlagert. Dies bedeutet, dass Anforderungen mit alternativen Daten beantwortet werden müssen, die nachträglich durch die eigentlich angeforderten Daten ersetzt werden können. In [Wat94b] werden für Datenobjekte innerhalb einer hierarchisch verlinkten Struktur von Hyperobjekten Platzhalter (Future) generiert. Damit wird die Antwortzeit für die Anforderung von Web-Seiten erhöht. Durch Futures ersetzte Hyperobjekte können durch den Nutzer über einen Verweis nachträglich angefordert werden. Eine weitere Reduzierung der Datenmenge wird durch eine feingranulare Strategie zum Laden von Web-Seiten erreicht. Für eine angeforderte Seite wird zunächst nur der auf dem Display darstellbare Bereich geladen. Weitere Teile einer Seite werden durch die Navigation des Benutzers angefordert.

Verzögertes Zurückschreiben

Eine Entkopplung des Zeitpunktes des Zurückschreibens von Änderungen der Daten wird durch ein verzögertes Zurückscheiben ermöglicht. Damit können Daten z. B. zunächst lokal geändert werden, bis alle Operationen abgeschlossen sind (z. B. wenn die Operation close auf einer Datei ausgeführt wird). Danach wird die Summe aller Änderungen über das Netzwerk propagiert [Kis96, FoZ94]. Ein verfeinerter Mechanismus in CODA verteilt das Zurückschreiben von Daten gleichmäßig auf die verfügbare Bandbreite schmalbandiger Verbindungen [MES95].

3.2.3 Adaption der Anwendungsstruktur

Die nachfolgend beschriebenen Mechanismen zur Strukturadaption arbeiten auf der Ebene der Anwendungsarchitektur. Diese besteht entsprechend der Sichtweise von Architekturbeschreibungssprachen (siehe Abschnitt 2.5.4) aus Komponenten und Konnektoren. Die Komponenten können auf unterschiedlichen Rechnern innerhalb des verteilten Systems platziert werden. Die Konnektoren stellen die Verbindungen und damit die Kommunikationsbeziehungen zwischen den verteilten Komponenten her. Insbesondere sind von Adaptionsmechanismen dieser Ebene auch die Mechanismen zur Parameteradaption betroffen. So kann ein spezifisches Kommunikationsprotokoll eingesetzt werden, wenn der Netzzugang über eine bestimmte Technologie erfolgt (z. B. Bluetooth). Muss aber z. B. nach einem Ortswechsel eine andere Zugangstechnologie verwendet werden, kann eine Anpassung durch das Ersetzen des spezifischen Kommunikationsprotokolls durch ein passendes Protokoll, d. h. durch eine Strukturadaption erfolgen. Mechanismen zur Strukturadaption lassen sich in die drei Hauptklassen Komponenten, Verbindungen und Platzierung für Mechanismen zur Adaption der Anwendungsstruktur einordnen.

3.2.3.1 Platzierung

Mechanismen zur Platzierung bzw. dynamischen Änderung der Platzierung stellen in Bezug auf die Anwendungsstruktur die einfachste Art der Adaption dar. Die Komponenten selbst und deren Verknüpfungen werden nicht verändert[8]. Es ändert sich lediglich der Ort[9],

[8] Dies trifft nur auf der Betrachtungsebene der Architekturbeschreibung zu. Dynamische Änderungen der Platzierung müssen vom Laufzeitsystem unterstützt werden. Auf dieser Ebene müssen meist auch Kommunikationsverbindungen angepasst werden.

[9] Im Sinne von Agentensystemen (siehe Abschnitt 2.3.3).

an dem Komponenten ausgeführt werden. Grundsätzlich können Platzierungsentscheidungen zur Erzeugungszeit und während der Laufzeit getroffen werden. Dementsprechend können Mechanismen zur Festlegung des Erzeugungsortes und zur Migration unterschieden werden (siehe Abbildung 3-13).

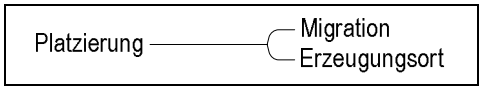


Abbildung 3-13: Mechanismen zur Adaption der Platzierung von Anwendungskomponenten

Erzeugungsort

Unter Mechanismen zur Festlegung des Erzeugungsortes von Komponenten werden Mechanismen eingeordnet, die vor dem Anwendungsstart eine Verteilung bzw. entfernte Erzeugung von Anwendungskomponenten ermöglichen. Ein bekannter Mechanismus ist das Laden von Applets auf den Client-Rechner einer Web-Anwendung. Dabei wird das Paradigma Code on Demand zur Erreichung der Codemobilität angewendet. Ein weiteres Paradigma, das die dynamische Verteilung von Anwendungskomponenten unterstützt, ist Remote Evaluation (siehe Abschnitt 2.3.3). Der Ansatz des Open Signaling für programmierbare Netzwerke stellt eine Plattform für die entfernte Codeinstallation zur Verfügung und zielt vorrangig auf die Installation von Code zur adaptiven Vermittlung von Paketen (siehe Abschnitt 2.3.3).

Migration

Mechanismen zur Migration ermöglichen eine dynamische Änderung des Ausführungsortes von Komponenten zur Laufzeit. Das Paradigma der mobilen Agenten unterstützt eine dynamische Änderung der Platzierung von Komponenten zur Laufzeit. Unterschieden werden kann hier zwischen starker und schwacher Mobilität (siehe Abschnitt 2.3.3). Eine weitere Form der dynamischen Platzierung wird durch programmierbare Netzwerke unterstützt (siehe Abschnitt 2.3.4).

3.2.3.2 Verbindungen

Verbindungen definieren die Kommunikationsbeziehungen zwischen den einzelnen Komponenten einer Anwendung und legen damit die an der Verarbeitung beteiligten Komponenten und deren Reihenfolge innerhalb der Anwendung fest. Durch die Änderung von Kommunikationsbeziehungen kann die Verarbeitungsreihenfolge und bei Verfügbarkeit alternativer Pfade auch die Funktionalität der Anwendung verändert werden, ohne die Komponenten und deren Platzierung zu ändern. Mechanismen zur Adaption der Kommunikationsbeziehungen sind das dynamische Binden, die Verwendung indirekter Verbindungen, das Verzweigen, Parallelisieren (Multicast) und das Vereinigen von Verbindungen (siehe Abbildung 3-14).

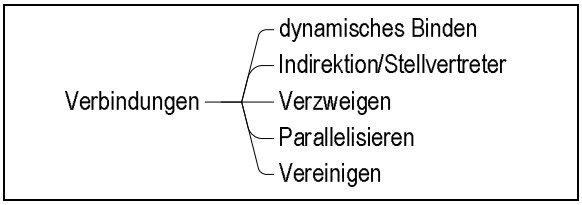


Abbildung 3-14: Mechanismen zur Adaption von Kommunikationsbeziehungen innerhalb einer Anwendungsarchitektur

Dynamisches Binden

Durch Mechanismen zum dynamischen Binden können Kommunikationsbeziehungen zur Laufzeit abgebrochen und, zwischen möglicherweise neuen Partnern, neu aufgebaut werden. Diese Mechanismen stellen somit eine wesentliche Grundlage für die Adaption der Anwendungsstruktur dar. Unterstützt wird ein dynamisches Binden unter anderem durch RPC-Realisierungen (siehe Abschnitt 2.1.2.3).

Verzweigung

Unter Verzweigung werden Mechanismen verstanden, die ausgehend von einem Kommunikationsport mehrere alternative Verbindungen aufbauen und Daten über eine dieser Alternativen weiterleiten. Damit erfolgt die Auswahl einer Kommunikationsverbindung. Genutzt werden können diese Mechanismen in Kombination mit der Replikation von Komponenten beispielsweise zur Lastverteilung.

Parallelisierung

Mechanismen zur Parallelisierung bauen, ausgehend von einem Kommunikationsport, mehrere Verbindungen auf und leiten Anwendungsdaten über alle diese Verbindungen an mehrere Empfänger weiter. Damit werden eintreffende Daten vervielfältigt und parallel an mehrere Empfänger weitergeleitet. In Verbindung mit der Replikation von Komponenten kann diese Art der Verzweigung beispielsweise zur fehlertoleranten Verarbeitung verwendet werden. Außerdem ist eine parallele Anwendung unterschiedlicher Operationen auf gleichen Daten möglich, wenn die Operationen nicht voneinander abhängen.

Vereinigung

In die Klasse Vereinigung werden Mechanismen eingeordnet, die mehrere eingehende Kommunikationsverbindungen auf eine ausgehende Kommunikationsverbindung abbilden. Damit werden eintreffende Daten gesammelt, evtl. verknüpft oder synchronisiert und dann über eine ausgehende Kommunikationsverbindung weitergeleitet. Mit einem solchen Mechanismus können beispielsweise zuvor parallelisierte Datenströme zusammengefasst bzw. synchronisiert werden.

Indirekte Kommunikation

Indirekte Kommunikationsverbindungen entkoppeln die Kommunikationspartner entsprechend des Stellvertreterkonzeptes (siehe Abschnitt 2.3.1). Damit können direkte Kommunikationsverbindungen aufgetrennt und weitere Komponenten zwischen die kommunizierenden Komponenten eingefügt werden. Die Indirektion bleibt dabei für die ursprünglich kommunizierenden Komponenten transparent. Beispielsweise können Mechanismen zur

Filterung oder zur Realisierung abgekoppelte Operationen in eine Anwendung eingefügt werden, ohne die Anwendungskomponenten zu verändern [Zen99, FiG94].

3.2.3.3 Komponenten

Komponenten enthalten sowohl reine Anwendungsfunktionalität als auch Adaptionsmechanismen. Durch Veränderungen auf der Ebene der Komponenten kann die Funktionalität der Anwendung adaptiert werden. Komponenten können entfernt, hinzugefügt, ausgetauscht und repliziert werden. Je nach Mechanismus sind von diesen Änderungen auch die Verbindungen zwischen den Komponenten betroffen. Die identifizierten Klassen von Mechanismen werden in Abbildung 3-15 aufgelistet.

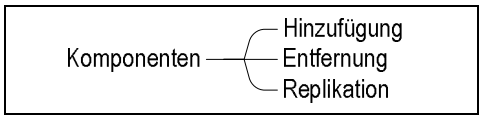


Abbildung 3-15: Mechanismen zur Adaption der Komponenten innerhalb einer Anwendungsstruktur

Hinzufügen

Architekturen können durch das Hinzufügen von Komponenten um Verarbeitungsschritte erweitert werden. Dies bewirkt auch eine Änderung der bestehenden Verbindungen zwischen Komponenten. Durch die Anwendung von Mechanismen zur indirekten Kommunikation können Komponenten auf einfache Weise zwischen bestehende Komponenten eingefügt werden (siehe Abschnitt 2.3.1). Eine weitere Möglichkeit zum Hinzufügen von Verarbeitungsschritten bietet das Konzept der Interceptoren [OMG01].

Entfernen

Das ersatzlose Entfernen von Komponenten reduziert die Anzahl der Verarbeitungsschritte innerhalb einer Architektur. Auch diese Mechanismen bewirken eine Veränderung der bestehenden Verbindungen zwischen Komponenten. Die Mechanismen zum Entfernen und Hinzufügen von Komponenten ermöglichen es insbesondere auch, Komponenten in einem Pfad auszutauschen bzw. umzuordnen. Beispielsweise kann eine Komponente zur Bildkonvertierung durch eine Komponente ersetzt werden, die einen alternativen Algorithmus bereitstellt. Auf Implementierungsebene kann das Ersetzen von Algorithmen auch innerhalb einer Komponente erfolgen und könnte damit als eigenständiger Mechanismus betrachtet werden. Auf Komponentenebene kann das Ersetzen jedoch durch das Entfernen und Hinzufügen von Komponenten erreicht werden. Dies führt insbesondere zu einer höheren Transparenz für die Adaption. Deshalb werden Ersetzen und Umordnen in der Klassifikation nicht explizit betrachtet.

Replizieren

Durch Mechanismen zur Replikation von Komponenten steht der Anwendung die Funktionalität der Komponente mehrfach zur Verfügung. Die Integration der replizierten Komponente erfordert Änderungen der Verbindungen der bestehenden Anwendung. In Kombination mit Mechanismen zur Verzweigung kann beispielsweise eine Lastverteilung erreicht werden.

3.2.4 Zusammenfassung

Das vorgeschlagene Schema zur Klassifikation von Basismechanismen zur Adaption betrachtet Adaption fein-granular entsprechend der eingesetzten Mechanismen zur Adaption. Die Einordnung erfolgt anhand der Bestandteile verteilter Anwendungen, die durch die Mechanismen adaptiert werden, d. h. anhand des Gegenstandes der Adaption. Das Klassifikationsschema gibt einen vollständigen Überblick über die Mechanismen zur Adaption und erlaubt die Einordnung der in Kapitel 2 vorgestellten Forschungsobjekte. Eine zusammenfassende Darstellung aller definierten Klassen von Adaptionsmechanismen ist in Abbildung 3-16 enthalten.

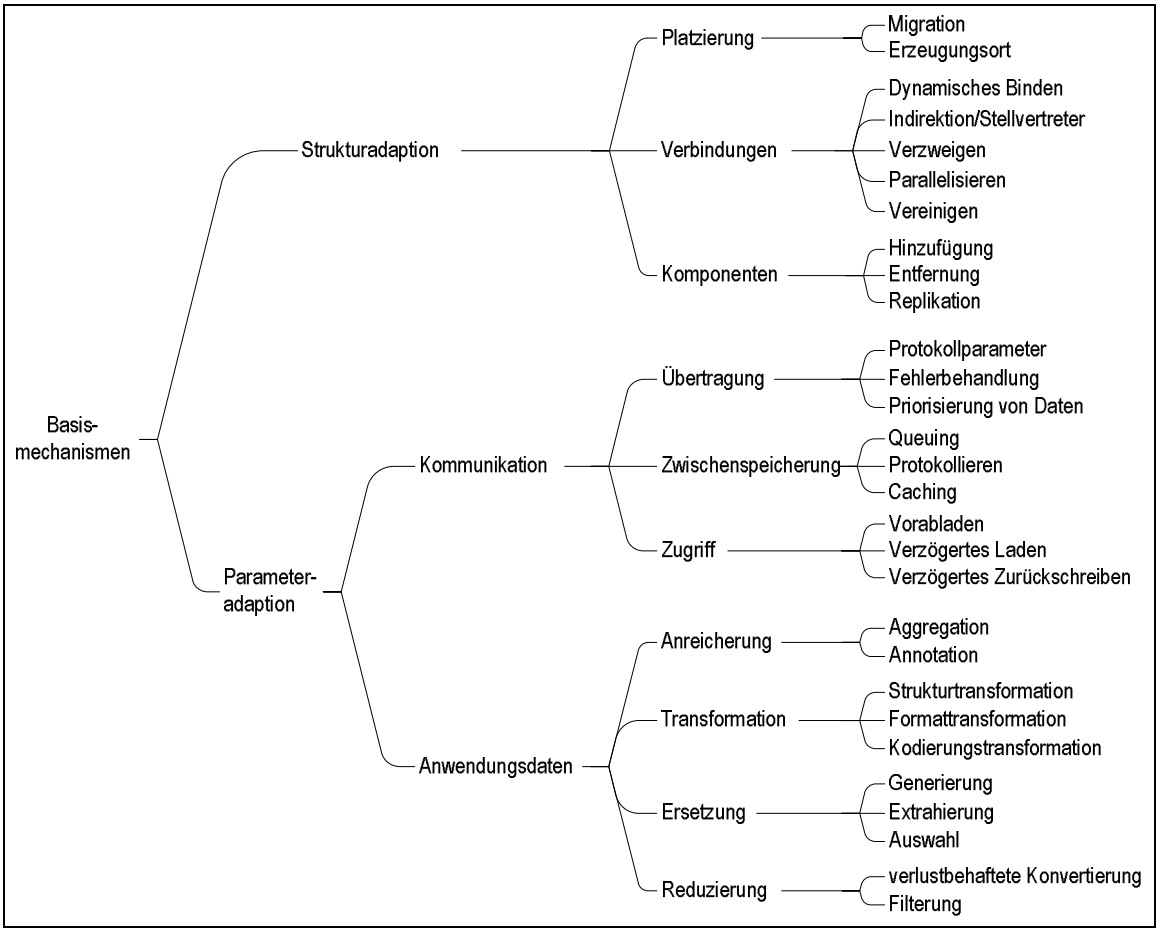


Abbildung 3-16: Zusammenfassende Darstellung der Basismechanismen zur Adaption

In Abschnitt 3.3 werden die Mechanismen zur Parameteradaption weiter untersucht. Dazu werden Bewertungskriterien erarbeitet, die sich auf die in Kapitel 1 genannten Anforderungen A1.1 bis A1.4 beziehen und den Einsatz der Mechanismen bezüglich der Anforderungen näher betrachten.

3.3 Analyse von Basismechanismen zur Adaption

Im vorangegangenen Abschnitt wurden Basismechanismen zur Adaption klassifiziert und anhand von Beispielen aus Forschungsprojekten veranschaulicht. Nachfolgend sollen die Mechanismen zur Parameteradaption in Bezug auf ihren Einsatz in adaptiven verteilten Anwendungen bewertet werden. Insbesondere sollen die Anforderungen A1.1 bis A1.4 aus Kapitel 1 aufgegriffen werden, um die Mechanismen in Bezug auf diese zu untersuchen.

Zunächst wird ein Bewertungsschema entwickelt, auf dessen Basis die Mechanismen abschließend vergleichend gegenübergestellt werden.

3.3.1 Bewertungsschema

Die Klassifikation anhand des Adaptionsgegenstandes gibt einen Überblick über die verfügbaren Klassen von Adaptionsmechanismen. Insbesondere wird eine Unterscheidung in Parameteradaption und Strukturadaption vorgenommen. Während Mechanismen zur Strukturadaption die eingesetzten Mechanismen, deren Platzierung und Verbindungen anpassen, beinhalten Mechanismen zur Parameteradaption die Anpassung von Anwendungsdaten und deren Übertragung. Das Bewertungsschema soll einen Vergleich der Mechanismen zur Parameteradaption hinsichtlich ihres Einsatzes und ihrer Auswirkungen ermöglichen. Dazu werden die folgenden Bewertungskriterien herangezogen.

3.3.1.1 Auswirkungen

Dieses Kriterium fasst die primären Auswirkungen der einzelnen Adaptionsmechanismen zusammen. Es wird untersucht, welche Veränderungen am Gegenstand der Adaption aus der Anwendung des jeweiligen Mechanismus' resultieren.

3.3.1.2 Anforderungen

Unter diesem Punkt werden die Anforderungen benannt, die ein Mechanismus an seine Ausführungsumgebung stellt. Diese umfassen unter anderem die benötigten Ressourcen bzw. ob weitere Adaptionsmechanismen notwendig sind, um einen Mechanismus einsetzen zu können.

3.3.1.3 Einsatz

Die Einsetzbarkeit bewertet, für welche Problemstellungen heterogener und dynamischer Infrastrukturen die einzelnen Mechanismen als Lösungsansatz verwendet werden können. Dabei kann Bezug auf die Auswirkungen genommen werden. Reduziert beispielsweise ein Mechanismus die Größe von Anwendungsdaten, ergeben sich daraus unter anderem Möglichkeiten zur Reduzierung der zu übertragenden Daten über eine schmalbandige Netzwerkverbindung oder zur Unterstützung von Endgeräten mit begrenztem Hauptspeicher. Nachfolgend soll ausgehend von den Anforderungen A1.1, A1.2 und A1.4 aus Kapitel 1 ein Schema zur Bewertung des Einsatzes der Adaptionsmechanismen entwickelt werden. Die Anforderungen werden dabei auf Einsatzbereiche (E) abgebildet, die nummeriert und in Teilbereiche untergliedert werden. Entsprechend dieser Untergliederung kann nachfolgend fein-granular der Einsatz von Mechanismen bestimmt werden.

Heterogene Anforderungen der Benutzer und der Anwendungssituation (A1.3) werden auf Benutzer- bzw. Anwendungsebene definiert und müssen auf technische Anforderungen abgebildet werden. Konkret können die Anforderungen der Benutzer und der Anwendungssituation auf die Einsatzbereiche E1 bis E3 abgebildet werden, die durch die technischen Anforderungen definiert werden.

Einsatzbereiche von Adaptionsmechanismen:

- E1 - Heterogene und limitierte Ressourcen der Endgeräte
  - E1.1 – geringe Rechenleistung
  - E1.2 – geringer Speicher
  - E1.3. - unterstützte Multimediaformate
  - E1.4 – kleine Display
  - E1.5 – spezifische Eingabe- und Ausgabegeräte, Navigation
- E2 – Heterogene und limitierte Qualität der Übertragungsmedien
  - E2.1 - geringe Datenrate, hohe Kosten
  - E2.2 – hohe Verzögerung, hohe Antwortzeit
  - E2.3 – hohe Fehleranfälligkeit
- E3 – Dynamik durch Mobilität
  - E3.1 - Verbindungsunterbrechungen
  - E3.2 – Phasen der Abkopplung
  - E3.3 – Variierendes Dienstangebot

3.3.1.4 Platzierung

Die Kategorie Platzierung beschreibt, an welchen Ausführungsorten die Mechanismen potentiell am effizientesten eingesetzt werden können. Unterschieden werden die Ausführungsorte Endgerät, Proxy im Festnetz und Server im Festnetz.

3.3.1.5 Anzahl der Komponenten

Das Kriterium Anzahl der Komponenten untersucht, wie viele Komponenten ein Adaptionsmechanismus umfasst. Beispielsweise bestehen verlustbehaftete Mechanismen wie Filterung und Konvertierung in der Regel aus einer Komponente, während verlustfreie Mechanismen meist zwei Komponenten umfassen. So müssen Daten nach einer Kompression vor einer weiteren Verarbeitung wieder dekomprimiert werden. Auch Mechanismen zur Übertragung bestehen in der Regel aus zwei Komponenten zum Senden und zum Empfangen der Daten.

3.3.2 Bewertung der Mechanismen zur Parameteradaption

Die nachfolgende Tabelle 3-1 enthält die Bewertung der Basismechanismen.

Mechanismus	Auswirkung	Anforderungen	Einsatz	Platzierung	Anzahl Komponenten
Filterung	reduziert Informationsgehalt, verwirft nicht unterstützte Datentypen und -formate	Filterausdruck	E2.1, E2.2, E1.3	Proxy, Server, (vor Übertragung)	eine
Konvertierung	reduziert Informationsgehalt, passt Dateninhalte an	Datenformat muss unterstützt werden	E1.4, E2.1, E2.2	Proxy, Server, (vor Übertragung)	eine
Generierung	erzeugt alternative Daten mit gleichem oder geringerem Informationsgehalt	spezifische Dateneigenschaften	E1.3, E1.5, E2.1, E2.2	Proxy, Server	eine
Extrahierung	extrahiert alternative Daten	Kenntnis über Gewinnung der alternativen Daten	E1.3, E2.1	Proxy, Server	eine
Auswahl	wählt passende Daten aus	Verfügbarkeit alternativer Daten	E1.3, E2.1	Proxy, Server,	eine
Kodierungstransformation	ändert Datenkodierung, reduziert Datenmenge (Kompression)	Rechenleistung, Kompression erfordert Rechenleistung auch auf Endgerät	E1.1, E2.1	Endgerät und Festnetz[10]	zwei
Formattransformation	ändert Datenformat	Unterstützung von Quell- und Zielformat	E1.1, E1.3	Endgerät und Festnetz[10]	zwei
Strukturtransformation	ändert Datenstruktur	Unterstützung von Quell- und Zielstruktur	E1.3, E1.4, E1.5	Endgerät und Festnetz[10]	zwei
Aggregation	erhöht Informationsgehalt, fasst Informationen verschiedener Ausgangsdaten zusammen, dadurch Datenreduzierung	Verfügbarkeit zusätzlicher Informationen, Definition von Aggregationsregeln	E1.2, E1.1, E2.1	Festnetz, vor Übertragung zur Reduzierung der Datengröße	eine
Annotation	erhöht Informationsgehalt, erhöht die Datenmenge, Meta-Informationen unterstützen Adaption	Verfügbarkeit zusätzlicher Informationen über Semantik der Daten, Regeln für die Generierung von Annotationen, Heuristiken	E1.4, E1.5	Festnetz (Semantik bzw. Struktur bekannt), Proxy (Heuristiken) evtl. Endgerät	eine
Verzögertes Zurückschreiben	verringert Zahl der Nachrichten über Netz, verteilt Netzlast über Zeit, erhöht Wahrscheinlichkeit von Konflikten	Speicher auf Endgerät, Kenntnis über ausgeführte Operationen, Mechanismen zur Konfliktauflösung	E2.1	Endgerät[11]	eine
Verzögertes Laden/ Verarbeiten	reduziert zu übertragende Datenmenge	Rechenleistung, Erstellung von Platzhaltern, Daten müssen explizit nachgeladen werden	E2.1, E2.2	Endgerät[11]	eine

[10] zwei Komponenten zur Kodierung/Dekodierung erforderlich, bei Dienstnutzung in der Regel Endgerät und Festnetz, je nach Platzierung des Dienstes

[11] Ort der lokalen Verarbeitung

Mechanismus	Auswirkung	Anforderungen	Einsatz	Platzierung	Anzahl Komponenten
Vorabladen	reduziert Antwortzeit, erhöht lokale Verfügbarkeit, nutzt nicht verwendete Netzkapazität	Speicher auf Endgerät, Bandbreite, Regeln zur Auswahl der Daten	E2.2	Endgerät, Server[11]	eine
Caching	reduziert Antwortzeit, reduziert zu übertragende Datenmenge, erhöht lokale Verfügbarkeit	Speicher auf Endgerät, Strategie zur Verdrängung von Datenobjekten aus Cache	E2.1, E2.2	Endgerät, Proxy[11]	eine
Protokollieren	erhöht Autonomie, speichert Daten zwischen, erhöht Antwortzeit	Speicher auf Endgerät, evtl. Optimierungsregeln	E3.1, E3.2	Endgerät[11]	eine
Queuing	erhöht Autonomie, speichert Daten zwischen, erhöht Antwortzeit	Speicher, Prioritätsvergabe	E2.1, E3.1, E3.2	Endgerät[11]	eine
Protokollparameter	bessere Ausnutzung der verfügbaren Bandbreite, reduziert Antwortzeit, verhindert Überlastung des Empfängers	erfordert Protokolländerungen bei Sender und Empfänger	E2	Endgerät und Festnetz[12]	zwei
Fehlerbehandlung, FEC	Fehlerkorrektur, kein Rückkanal erforderlich, keine Übertragungswiederholung bei Korrektur	erhöht Datenmenge durch Redundanz	E2.3	Endgerät und Festnetz[12]	zwei
Fehlerbehandlung, ARQ	Fehlererkennung und Übertragungswiederholung	Rückkanal erforderlich, hohe Übertragungszeiten im Fehlerfall	E2.3	Endgerät und Festnetz[12]	zwei
Priorisierung von Daten	verschiedene Dienstklassen für Daten, optimierte Nutzung von Verbindungen, hohe Antwortzeiten für Daten mit geringer Priorität	Verwaltungskomponente und Warteschlangen für unterschiedliche Prioritäten, Vergabe der Prioritäten	E2.1, E2.2, E3.1, E3.2	Sender	eine

Tabelle 3-1: Bewertung der Basismechanismen

3.4 Fazit

Im vorliegenden Kapitel wurden wesentliche Basismechanismen für die Adaption mobiler und ubiquitärer Anwendungen identifiziert. Dafür wurden die in Kapitel 2 analysierten Lösungsansätze in ihre Grundbausteine zerlegt. Die Feststellung wesentlicher Gemeinsamkeiten vieler Lösungen hinsichtlich der verwendeten Mechanismen führte zur Identifikation von Basismechanismen. Zur systematischen Betrachtung dieser wurde im Rahmen der Arbeit ein Klassifikationsschema entwickelt. Dieses ermöglicht die eindeutige Einordnung aller identifizierten Basismechanismen. Das Klassifikationsschema ermöglicht außerdem eine systematische Betrachtung und Einordnung der existierenden Lösungsansätze zur Adaption. Insbesondere konnten weitere Basismechanismen durch Lücken in der Klassifi-

[12] Auf Sender und Empfänger, bei Dienstnutzung in der Regel Endgerät und Festnetz, je nach Platzierung des Dienstes

kation und eine systematischen Suche nach Lösungsansätzen für die nicht belegten Klassen gefunden werden.

Außerdem erfolgte eine Bewertung der Basismechanismen hinsichtlicht der Anforderungen A1.1 bis A1.4 aus Kapitel 1. Festgestellt wurde, dass für alle Anforderungen Mechanismen verfügbar sind. In existierenden Lösungsansätzen wurden diese zum Teil kombiniert. So realisiert das verteilte Dateisystem CODA abgekoppelte Operationen, die aus den Basismechanismen Vorabladen, Caching, Protokollieren einer lokalen Verarbeitung und einem Verzögerten Zurückschreiben der Daten realisiert werden. Diese behandeln vor allem längere Phasen der Abkopplung. In Rover werden ebenfalls abgekoppelte Operationen betrachtet, diese aber mit Queuing für RPC Nachrichten kombiniert, um unzuverlässige Kommunikationskanäle zu unterstützen. Orthogonal dazu können Mechanismen zur Datenadaption eingesetzt werden, um Eigenschaften des Endgerätes und begrenzte Kommunikationsressourcen zu unterstützen. Die daraus abgeleitete Lösungsidee der Komposition von Basismechanismen zur systematischen Entwicklung adaptiver Anwendungen wird im folgenden Kapitel mit der Erarbeitung des Meta-Modells vertieft. Insbesondere sollen der Zugriff auf Informationen über die Ausführungsumgebung sowie verfügbare Meta-Informationen in die Modellierung einbezogen werden, um Adaptionsmechanismen gezielt steuern zu können.

Kapitel 4

EIN KOMPONENTENBASIERTES META-MODELL ZUR BESCHREIBUNG KONTEXTABHÄNGIGER ADAPTIONSGRAPHEN

4.1 Zielstellung der Modellierung

Im vorliegenden Kapitel wird ein Meta-Modell zur Beschreibung adaptiver Anwendungen erarbeitet. Instanzen des Meta-Modells sind Anwendungsmodelle, mit dem Entwickler Anwendungen aus der Sicht der Adaption in Form gerichteter Graphen beschreiben können. Diese Sicht kann mit weiteren Sichten kombiniert werden, um vollständige Anwendungen zu modellieren.

Das Meta-Modell soll die wesentlichen Elemente definieren, auf deren Basis adaptive Anwendungen fein-granular beschrieben werden können. Die Grundlage der Beschreibung bilden die Basismechanismen, die in Kapitel 3 identifiziert, klassifiziert und bewertet wurden. Insbesondere sollen Parameter- und Strukturadaption integriert betrachtet werden. Bei der Modellierung wird eine Wiederverwendung sowohl einzelner Mechanismen als auch von Kombinationen von Mechanismen angestrebt. Ein weiterer wesentlicher Aspekt ist die explizite Beschreibung der Steuerung der Adaption durch Kontext. Es können die folgenden Anforderungen an das zu entwickelnde Meta-Modell formuliert werden:

A4.1. *Integration:* Das Meta-Modell soll die Mechanismen zur Struktur- und Parameteradaption integrieren. Es soll insbesondere keine Spezialisierung auf bestimmte Anwendungsszenarien oder Adaptionsmechanismen erfolgen.

A4.2. *Flexibilität:* Das Meta-Modell soll eine flexible Komposition von Adaptionsmechanismen unterstützen. Die Adaptionsmechanismen sollen als Grundbausteine modelliert werden, die sowohl wiederverwendbar als auch ersetzbar sind.

A4.3. *Transparenz:* Der Aspekt der Adaption soll transparent beschrieben werden. Dies umfasst die genaue Definition der eingesetzten Adaptionsmechanismen und die explizite Beschreibung der verwendeten Kontextwerte sowie der Verwendung dieser Information zur Steuerung der Adaption.

A4.4. *Plattformunabhängigkeit*: Das Meta-Modell soll unabhängig von bestimmten Implementierungsplattformen sein, d. h. beschriebene Anwendungen sollen auf unterschiedlichen Geräte- und Komponentenplattformen mit Hilfe unterschiedlicher Implementierungssprachen realisiert werden können und damit der Heterogenität und Dynamik der Zielinfrastruktur (Kapitel 1, Anforderungen A1.1 bis A1.4) Rechnung tragen.

4.2 Vorbetrachtungen

Der Erarbeitung des Meta-Modells sollen einige Vorbetrachtungen vorangestellt werden. Zunächst werden die grundlegenden Bestandteile adaptiver Anwendungen beschrieben, die eine Basisstruktur für das Meta-Modell festlegen. Danach werden existierende Ansätze (siehe Abschnitt 2.5) betrachtet, die eine Grundlage für das Meta-Modell darstellen bzw. mit denen das Meta-Modell in enger Beziehung steht.

4.2.1 Grundlegende Bestandteile adaptiver Anwendungen

Ausgehend von den Erkenntnissen aus Kapitel 2 und 3 können die grundlegenden Bausteine adaptiver Anwendungen identifiziert werden (siehe Abbildung 4-1). Adaptive Anwendungen bestehen aus Anwendungskomponenten, welche die Anwendungslogik und die Adaptionsmechanismen enthalten, einer Laufzeitumgebung sowie Anwendungsdaten. Außerdem stehen mit Kontext Informationen über die Ausführungsumgebung zur Verfügung. Die *Adaptionsmechanismen* führen die Strukturadaption sowie die zur Adaption der Anwendungsdaten und deren Übertragung notwendigen Verarbeitungsschritte aus. Die *Laufzeitumgebung* stellt den Adaptionsmechanismen im Sinne eines Containers (siehe Abschnitt 2.5.2) Basisdienste wie Naming, Persistenz, Transaktionsmanagement und die Verwaltung des Lebenszyklus' von Anwendungskomponenten zur Verfügung. Außerdem muss durch diese die Steuerung der Adaption erfolgen. *Kontext* repräsentiert Informationen über die Ausführungsumgebung, die zur Steuerung der Adaption verwendet werden können (siehe Abschnitt 2.4). Die *Anwendungsdaten* sind einer der Gegenstände der Adaption. Sie können um Meta-Informationen zur Unterstützung der Adaptionsverarbeitung (siehe Abschnitt 2.1.3.5) erweitert werden. Diese Meta-Informationen müssen explizit vom Anwendungsentwickler oder Mediengestalter zu den eigentlichen Mediendaten hinzugefügt werden und unterscheiden sich damit grundlegend von Kontext. Beispielsweise können für ein Bild Informationen über dessen Inhalt zur Verfügung gestellt werden (z. B. Bildtyp ist geographische Karte). Diese Meta-Informationen können dann bei der Adaption berücksichtigt werden (z. B. die Karte wird nicht skaliert, sondern in Segmente zerlegt, die in der ursprünglichen Größe angezeigt werden).

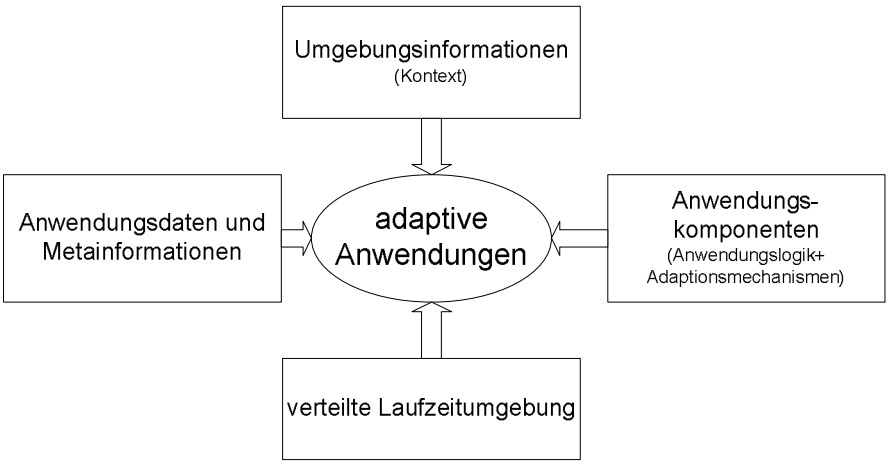


Abbildung 4-1: Bausteine für adaptive Anwendungen in verteilten Systemen

Der Zusammenhang zwischen den Grundbausteinen wird in Abbildung 4-2 veranschaulicht. Verteilt auf unterschiedlichen Rechnern innerhalb der Festnetzinfrastruktur und auf (mobilen) Endgeräten befindet sich eine Laufzeitplattform, die den Anwendungskomponenten Basisdienste für deren Abarbeitung zur Verfügung stellt. Der Kontextdienst

ist für adaptive Anwendungen ein wesentlicher Basisdienst und steht den Anwendungskomponenten zur Verfügung. Die Adaptionskomponenten befinden sich als Teilmenge der Anwendungskomponenten verteilt auf den Rechnern innerhalb der Infrastruktur. Ihr Lebenszyklus wird von einem lokalen Container gesteuert. Außerdem stehen ihnen über Schnittstellen des Containers Basisdienste zur Verfügung. Anwendungsdaten werden von der verteilten Anwendung zwischen Komponenten ausgetauscht. Der Austausch kann lokal oder über das Netzwerk erfolgen.

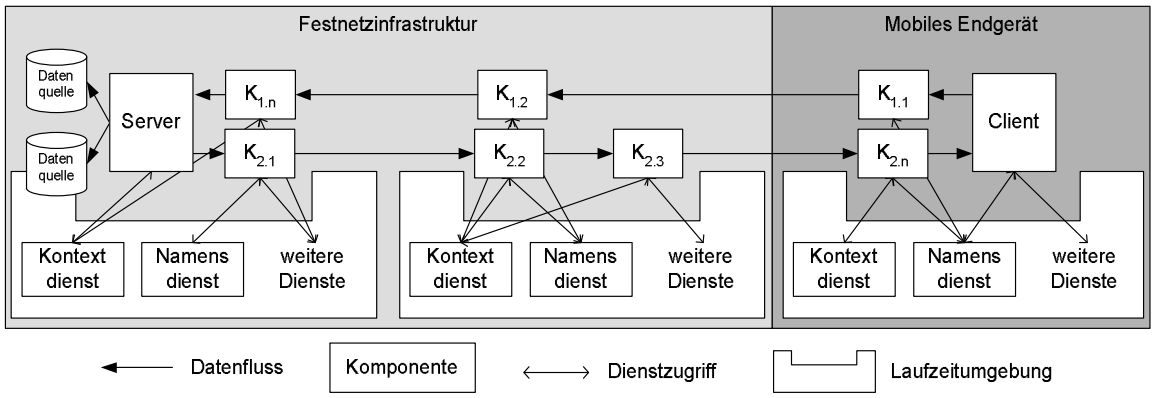


Abbildung 4-2: Zusammenhang zwischen den Grundbausteinen adaptiver Anwendungen in einer verteilten, mobilen Infrastruktur am Beispiel einer verteilten Client/Server-Anwendung

Für die Entwicklung des Meta-Modells wird angenommen, dass ein verteilter Kontextdienst zur Verfügung steht, der alle angeforderten Kontextwerte liefern kann. Von konkreten Laufzeitumgebungen soll abstrahiert werden, um eine plattformunabhängige Beschreibung adaptiver Anwendungen zu ermöglichen. Beschreibungsgegenstände des Meta-Modells sind somit die Adaptionskomponenten sowie deren Steuerung durch Kontext und Meta-Informationen. Anwendungskomponenten, die keine Adaptionsmechanismen enthalten, sollen ebenfalls in das Meta-Modell integriert werden können, um deren Interaktionen mit Adaptionskomponenten beschreiben zu können.

4.2.2 Grundlagen des Meta-Modells

Die Anforderungen A4.1 bis A4.4 haben sowohl auf die Ebene der Modellierung als auch auf die Elemente des Meta-Modells Auswirkungen. Die Anforderung A4.4 trägt vor allem der Heterogenität Rechnung und fordert eine Unabhängigkeit von Geräte-, Laufzeit- und Komponentenplattformen sowie einer bestimmten Implementierungssprache. Erfüllt das Modell die Anforderung der Plattformunabhängigkeit, können Anwendungsmodelle von Implementierungsdetails abstrahieren und anschließend auf verschiedenste Zielsysteme abgebildet werden. Diese Vorgehensweise entspricht dem Ansatz der Model Driven Architecture (siehe Abschnitt 2.5.4). Gemäß diesem Ansatz durchläuft der Softwareentwicklungsprozess mehrere Abstraktionsebenen vom Entwurf bis hin zum lauffähigen Code. Das zu entwickelnde Meta-Modell soll demzufolge die Beschreibung plattformunabhängiger Modelle ermöglichen. Die Plattformunabhängigkeit bezieht sich dabei auf die oben genannten Kriterien.

Die Anforderung A4.2 fordert eine größtmögliche Flexibilität bei der Komposition der Adaptionsmechanismen zu Adaptionsgraphen. Softwarekomponenten entsprechen dieser Forderung (Abschnitt 2.5.2). Sie stellen autonome Bausteine für die Softwareentwicklung dar und ermöglichen insbesondere die Ersetzbarkeit und Wiederverwendung einzelner Komponenten. Als Sicht auf Komponenten wird die der Komponentenspezifikation ge-

wählt (siehe [ChD00]), da diese mit der Forderung nach einem plattformunabhängigen Modell korrespondiert. Außerdem wird durch diese die Architektur in das Zentrum der Betrachtung gerückt. Dies entspricht dem Ziel der Modellierung, adaptive Anwendungen anhand der verwendeten Basismechanismen zu beschreiben. Die Basismechanismen können als Softwarekomponenten modelliert werden, die eine Komposition zu adaptiven Anwendungen ermöglichen.

Die Forderung nach Flexibilität wird auch durch das Strukturmuster Pipes-and-Filters erfüllt (siehe Abschnitt 2.1). Dieses beschreibt die grundlegende Architektur der betrachteten Anwendungsklasse und ermöglicht insbesondere eine lose Kopplung von Filtern bzw. Komponenten. Dies wird durch generische Schnittstellen zwischen den Komponenten erreicht. Das Meta-Modell greift deshalb wesentliche Elemente des Architekturmusters „Pipes und Filters“ auf, wodurch eine Beschränkung der beschreibbaren Architekturen auf die für adaptive Anwendungen wesentliche Struktur erreicht wird. „Pipes and Filters“ enthält jedoch einige Einschränkungen, die für das Meta-Modell zu restriktiv sind (z. B. Verbinden jeweils genau eines Readers und Writers, Datenaustausch nur zwischen benachbarten Filtern). Deshalb werden nachfolgend neben den Elementen zur Beschreibung der Zusammenhänge zwischen Kontext und Komponenten bzw. Konnektoren, weitere Elemente eingeführt, die diese Restriktionen überwinden.

Die Beschreibung von Software auf der Ebene der Architektur erfüllt sowohl die Forderung der Plattformunabhängigkeit als auch die der Flexibilität. Diese Beschreibungsebene abstrahiert von Implementierungsdetails einzelner Komponenten und beschreibt Softwaresysteme grob-granular anhand der Komponenten und deren Verbindungen (siehe Abschnitt 2.5.4). Dies ermöglicht eine Fokussierung des Modells auf die Kombination von Adaptionsmechanismen und die Beschreibung der Adaptionsentscheidungen. Gleichzeitig werden alle Details von Plattformen sowie einzelner Adaptionsmechanismen verborgen, die nicht Gegenstand der Modellierung sein sollen. Die grundlegenden Modellelemente von ADLs sind Komponenten, Konnektoren und Architekturen als Kombination dieser Elemente.

Der Anforderung A4.3 nach Transparenz kann damit zum Teil schon entsprochen werden. Adaptionsmechanismen werden in Form von Komponenten und Konnektoren explizit beschrieben. Da dies für alle Adaptionsmechanismen entsprechend der Klassifikation aus Kapitel 3 gilt, ist eine Integration aller Basismechanismen in die Modellierung gegeben (Anforderung A4.1). Um die Zusammenhänge zwischen Kontext und der Steuerung der Adaption beschreiben zu können sind jedoch Erweiterungen der Grundelemente von ADLs notwendig.

4.2.3 Auswahl einer Spezifikationssprache

In Abschnitt 2.5 wurden verschiedene Ansätze zum Entwurf und zur Beschreibung adaptiver Systeme betrachtet. Architekturbeschreibungssprachen unterstützen die Beschreibung von Softwaresystemen auf der Ebene der Architektur. Damit stellen ADLs eine mögliche Sprache für die Modellierung adaptiver Anwendungen dar. Es existiert jedoch keine Standardsprache, die in der Arbeit verwendet werden könnte.

Die Konzepte der OMG enthalten dagegen mit UML eine Standardsprache, die in der Softwareentwicklung eine weite Verbreitung gefunden hat. Im Vergleich zu ADLs bietet UML jedoch weit weniger Unterstützung für die Beschreibung von Softwarearchitekturen. MOF definiert eine abstrakte Sprache zur Spezifikation von Meta-Modellen. MOF und UML besitzen eine gemeinsame Untermenge von Modellelementen, die Struktur beider Sprachen ist sehr ähnlich. Als Teil der Spezifikation des Enterprise Distributed Object Computing (EDOC) wurde deshalb ein UML Profile für MOF definiert [OMG04], das es

ermöglicht, die graphische Notation sowie Werkzeuge für UML auch für die Spezifikation von MOF Meta-Modellen zu verwenden. Auf dieser Basis können Meta-Modelle unter Verwendung der graphischen Notation von UML beschrieben werden.

Für die Spezifikation des Meta-Modells adaptiver Anwendungen wird deshalb MOF in Verbindung mit UML ausgewählt. Eine Umsetzung des Meta-Modells in ein UML Profile ist damit später problemlos möglich. Aufgrund der fehlenden Unterstützung von UML Profilen in UML Entwicklungswerkzeugen (siehe z. B. [Ple02]) wird auf die Spezifikation eines UML Profiles im Rahmen der Arbeit verzichtet.

4.3 Entwicklung eines Meta-Modells zur Beschreibung von Adaptionsgraphen

Entsprechend der Vorüberlegungen sollen nachfolgend die Elemente des Meta-Modells und deren Beziehungen beschrieben werden. Die Spezifikation basiert auf MOF sowie der graphischen Notation entsprechend UML. Die einzelnen Elemente werden schrittweise eingeführt. Die Klassendiagramme enthalten deshalb jeweils Ausschnitte des Meta-Modells. In Anhang A sind Regeln zur Wohlgeformtheit des Meta-Modells enthalten, die Aspekte ergänzen, die nicht von den Klassendiagrammen erfasst werden. Außerdem enthält Anhang B eine Zusammenfassung der graphischen Notation des Meta-Modells.

4.3.1 Die Modellierung der Anwendungsdaten

Anwendungsdaten werden in einem Datencontainer gekapselt. Der Begriff Container wird in diesem Zusammenhang als Behälter der Daten sowie weiterer Informationen verstanden und ist von der Semantik eines Containers als Laufzeitplattform für Komponenten zu unterscheiden. Ein Datencontainer enthält Attribute zur Beschreibung des enthaltenen Datenobjektes, die Daten selbst sowie weitere, für die Adaption und Vermittlung innerhalb des Adaptionsgraphen notwendige, Meta-Informationen (siehe Abbildung 4-3). Meta-Informationen können durch Elemente der Klasse „Annotation“ in den Datencontainer eingefügt werden.

Im Modell werden einfache Datentypen für Parameter, Sensoren usw. sowie Typen von Datenobjekten, die Anwendungsdaten enthalten, unterschieden. Deshalb sollen letztere zur Unterscheidung von einfachen Datentypen nachfolgend als Objekttypen bezeichnet werden.

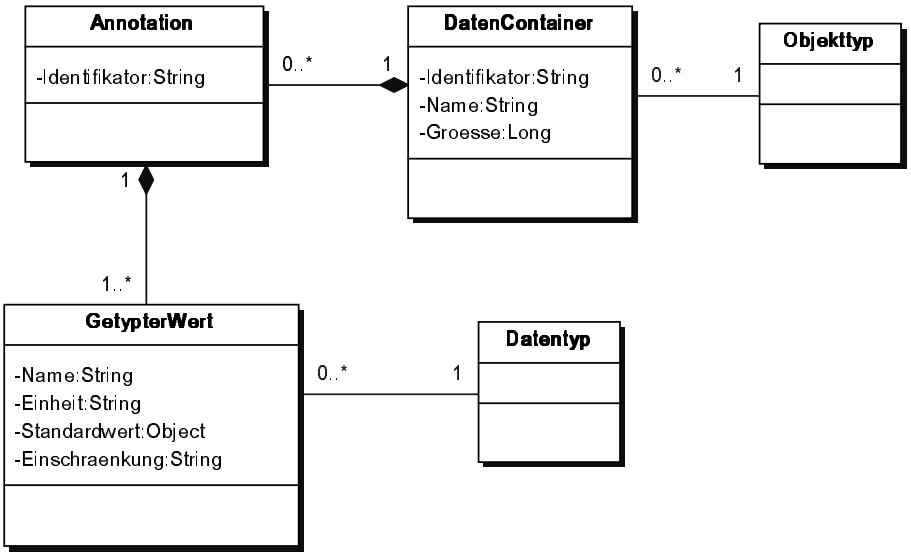


Abbildung 4-3: Das Klassendiagramm der Elemente zur Modellierung von Anwendungsdaten

4.3.1.1 Datentyp

Das Element „Datentyp“ repräsentiert das Konzept von Datentypen auf der Ebene des Meta-Modells (M2) und stellt einen Konstruktor von Datentypen dar. Auf Modellebene (M1) sind Instanzen von „Datentyp“ konkrete Datentypen. Das Meta-Modell definiert keine eigenen Datentypen sondern nimmt Bezug zu den Datentypen in MOF [OMG02c]. Demnach stehen im Modell (M1) die sechs Basistypen Boolean, Integer, String, Long, Float und Double sowie eine Menge von komplexen Datentypen zur Verfügung. Dies sind entsprechend des MOF Standards die Typen Enumeration, Structure, Collection und Alias.

4.3.1.2 GetypterWert

Das Element „GetypterWert“ repräsentiert im Meta-Modell Werte, die durch einen Namen, einen Datentyp, eine Einheit, einen optionalen Standardwert sowie optional eine Einschränkung des durch den Datentyp vorgegebenen Wertebereichs (z. B. die Einschränkung auf einen Wertebereich zwischen 0 und 100 für den Typ Integer) definiert werden. Die Assoziation zum Element „Datentyp“ drückt die Möglichkeit der Zuweisung eines Types auf Modellebene (M1) aus. Instanzen von „GetypterWert“ besitzen auf Modellebene (M1) ein weiteres Attribut, das einen Wert des definierten Typs aufnehmen kann. Ein getypter Wert kann auf Modellebene (M1) auf die folgende Weise notiert werden:

```
Wertname=(Typ, Einheit, Standardwert, Einschränkung).
```

Beispielsweise definiert der Ausdruck Schwellwert=(Integer, „Byte“, 50, „0-100“) einen Schwellwert vom Typ Integer mit der Einheit „Byte“, dessen Wertebereich auf Werte zwischen 0 und 100 eingeschränkt wird. Der Standardwert wird mit 50 definiert.

4.3.1.3 Annotation

Das Element „Annotation“ dient der Beschreibung von Meta-Informationen, die Datenobjekten während der Verarbeitung innerhalb des Adaptionsgraphen hinzugefügt werden können. Eine Annotation wird durch einen eindeutigen Identifikator gekennzeichnet und besteht aus mindestens einem getypten Wert (Element „GetypterWert“). Komponenten,

deren Verarbeitung nach dem Einfügen der Meta-Informationen stattfindet, können innerhalb ihrer Verarbeitungsoperationen auf diese Informationen zugreifen. Eine explizite Beschreibung dieses Zusammenhangs wird durch das Meta-Modell unterstützt. Detaillierte Erläuterungen dazu enthält Abschnitt 4.4.7. Eine Annotation wird entsprechend Abbildung 4-4 in ausführlicher (links) und verkürzter Schreibweise (rechts) notiert.

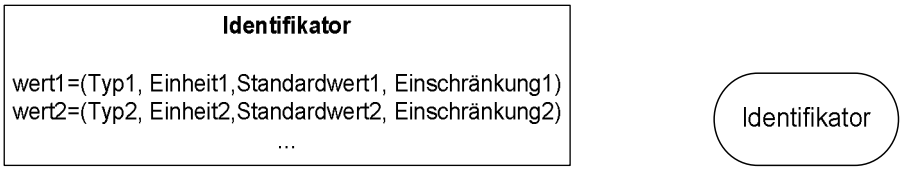


Abbildung 4-4: Notation einer Annotation in ausführlicher (links) und verkürzter Schreibweise (rechts)

4.3.1.4 Objekttyp

Das Element „Objekttyp“ repräsentiert im Meta-Modell die Klasse der Objekttypen (M2). Für Objekttypen gilt ähnliches wie für Datentypen. Auf Modellebene (M1) sind Instanzen von „Objekttyp“ konkrete Objekttypen. Im Meta-Modell wird das Element „Objekttyp“ vom primitiven Datentyp String abgeleitet, der in der Spezifikation von MOF definiert wird. Zeichenketten auf der Modellebene (M1) bezeichnen konkrete Objekttypen.

Grundlage der Beschreibung von Objekttypen in Form von Zeichenketten ist die MIME-Spezifikation [FrB96]. Diese definiert Medientypen durch die Elemente Typ und Subtyp. Ein Subtyp ist dabei genau einem Typ untergeordnet. Außerdem können weitere Parameter definiert werden, die zusätzliche Informationen zur Typbeschreibung liefern, z. B. die verwendete Kodierung der Zeichen eines Textes (Content-type: text/plain; charset=iso-8859-1). In [Mar00] wird ein ähnlicher Ansatz zur Beschreibung der Typen von Medienobjekten in Form von Signaturen verwendet. Dieser kann in ähnlicher Weise Typ- und Qualitätsinformationen enthalten. Neben dem Typ und Subtyp enthalten Signaturen noch eine weitere Unterstruktur von Subtyp, die Kodierung. Diese wird jeweils genau einem Subtypen zugeordnet (z. B. ([Typespec] Audio/Waveform/WAV [Quality] Channels=Stereo, Sampling_Frequency=44100; Sample_Depth=16)). Die Typbeschreibung einer Signatur besteht somit aus den drei Beschreibungselementen Typ, Subtyp und Kodierung, die eine vierstufige Typhierarchie beschreiben, wenn von einem gemeinsamen Wurzelelement ausgegangen wird, von dem alle Typen abgeleitet werden.

Für das Meta-Modell wird der Ansatz der drei Beschreibungsebenen von Signaturen aus [Mar00] für Objekttypen übernommen. Auf die Beschreibung von Qualitätsinformationen wird aus Gründen der Vereinfachung im Rahmen der Arbeit verzichtet. Die Notation von Objekttypen erfolgt in der folgenden Form:

```
Objekttyp=(Typ/Subtyp/Kodierung).
```

Ein Objekttyp wird immer durch diese drei Elemente definiert. Entsprechend der Betrachtung der Objekttypen als Typhierarchie mit vier Ebenen sind konkrete Objekttypen somit immer auf Ebene vier definiert. Existiert nur eine Kodierung für einen bestimmten Objekttyp, wird diese als „Standardkodierung“ bezeichnet. Zur Vereinfachung der Modellierung kann die Beschreibung von Objekttypen deshalb verkürzt angegeben werden. Beispielsweise bezeichnet der Objekttyp t_o=(image/gif) alle Bilder im GIF-Format. Existiert für diesen Typ nur eine Kodierung, bezeichnet der Objekttyp t_o somit den konkreten Objekttyp (image/gif/standardkodierung). Konkrete Datenobjekte (z. B. in einem Datencontainer)

müssen eindeutig definiert werden, d. h. der Objekttyp muss durch Typ, Subtyp und Kodierung definiert werden oder es existiert genau eine Kodierung so dass durch die Angabe von Typ und Subtyp ein eindeutiger Objekttyp definiert wird.

Signaturen von Ports und Rollen beschreiben dagegen eine Menge von Objekttypen (siehe Abschnitt 4.3.2.3). Zur Vereinfachung der Notation könen deshalb für Signaturen die folgenden verkürzten Schreibweisen verwendet werden:

Objekttyp=({Typ_1|Typ_2|…|Typ_n}),

Objekttyp=(Typ/{$Subtyp_1$|$Subtyp_2$|…|$Subtyp_n$}),

Objekttyp=(Typ/Subtyp/{$Kodierung_1$|$Kodierung_2$|…|$Kodierung_n$}).

Die erste Notation definiert dabei eine Menge von Objekttypen, die zu jedem angegebenen Typ alle Subtypen mit jeweils allen Kodierungen enthält. In ähnlicher Weise definierte die zweite Notation die Menge der Objekttypen des definierten Typs mit den angegebenen Subtypen und jeweils allen zugehörigen Kodierungen. Die dritte Notation definiert die Menge aller Objekttypen des definierten Typs und Subtyps mit allen angegebenen Kodierungen.

Objekttypen beschreiben die Typen von Datenobjekten getrennt von deren Typ in einer Programmier- bzw. Laufzeitumgebung. Beispielsweise kann ein Bild in Java als ByteArray oder Klasse der Java Advanced Imaging Bibliothek [Sun99] vorliegen. Der Objekttyp wird unabhängig davon im Adaptionsgraphen mit (image/gif/standardkodierung) beschrieben. Objekttypen sind damit unabhängig von bestimmten (plattformabhängigen) Typsystemen. Neue Typen von Datenobjekten können außerdem einfach in Form neuer Bezeichner hinzugefügt werden. Beispielsweise kann für Nachrichten zur Anforderung von Datenobjekten im Adaptionsgraphen ein Objekttyp (message/request/standardkodierung) mit weiteren Kodierungstypen definiert werden. Wird dieser Objekttyp von den Komponenten unterstützt, die diese Datenobjekte verarbeiten sollen, kann dieser Typ ohne weitere Änderungen von Komponenten oder Konnektoren verwendet werden. Es muss nur eine Definition im Typsystem erfolgen. Die Definition von Objekttypen ist damit flexibel und einfach erweiterbar (siehe auch [Mar01]).

4.3.1.5 DatenContainer

Datenobjekte werden im Meta-Modell durch einen Datencontainer beschrieben. Dieser enthält genau ein Datenobjekt, einen eindeutigen Identifikator für dieses Datenobjekt, dessen Namen, dessen Objekttyp und dessen Größe (in Byte). Außerdem kann ein Datencontainer keine, aber auch beliebig viele Annotationen enthalten. Durch die Assoziation mit dem Element „Objekttyp“ besitzt ein Datencontainer auf Modellebene (M1) ein weiteres Attribut, das einen Objekttyp aufnehmen kann. Auf dieses kann dann als explizite Repräsentation des Objekttyps der enthaltenen Daten zugegriffen werden.

4.3.2 Die Elemente zur Beschreibung der Struktur von Adaptionsgraphen

Begonnen wird mit einem Überblick über die Elemente zur Beschreibung der grundlegenden Struktur von Adaptionsgraphen. Diese sind in Abbildung 4-5 dargestellt. Ein Adaptionsgraph besteht demnach aus mindestens zwei Komponenten (einer Datenquelle und einer Datensenke) sowie mindestens einem Konnektor zur Verbindung dieser.

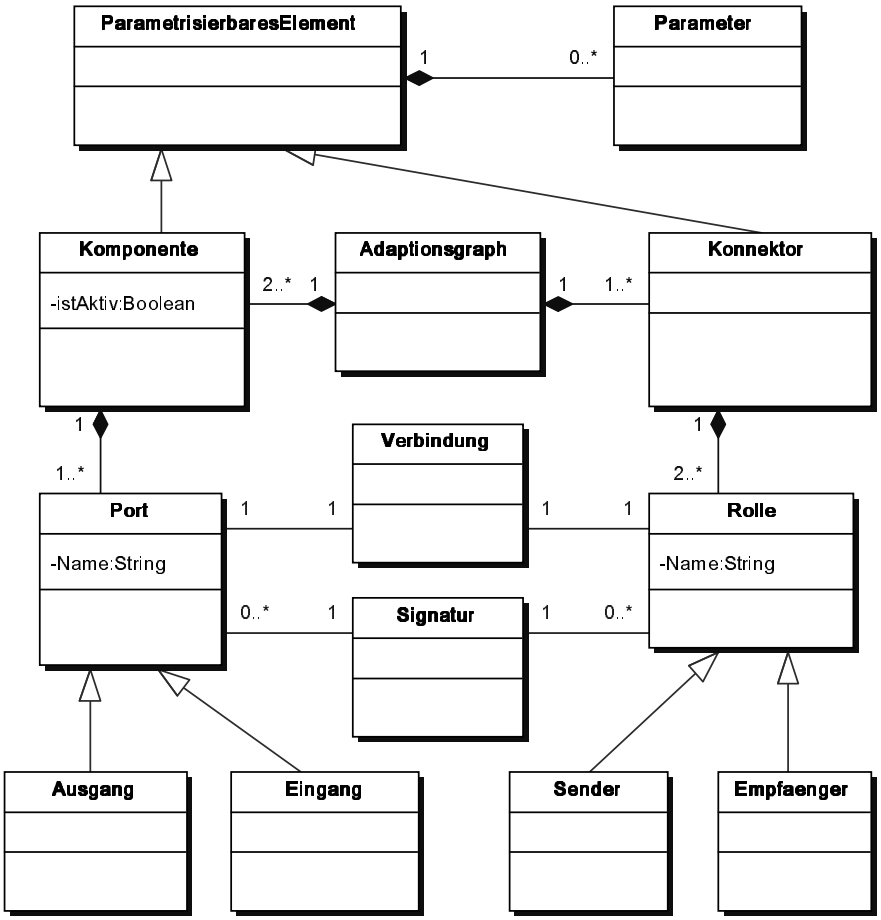


Abbildung 4-5: Das Klassendiagramm der Elemente zur Beschreibung von Adaptionsgraphen

Rollen und Ports definieren Interaktionsschnittstellen von Komponenten bzw. Konnektoren. Durch die Verknüpfung von Rollen und Ports werden Komponenten und Konnektoren zu Adaptionsgraphen kombiniert. Im Modell beschreibt ein Port bzw. eine Rolle eine Interaktionsschnittstelle zur Übergabe von Datenobjekten. Datenobjekte können entsprechend der Richtung des Datenflusses entweder empfangen oder versendet werden. Ports werden Komponenten zugeordnet. Jede Komponente besitzt mindestens einen Port. Entsprechend der Richtung des Datenflusses können Eingangs- und Ausgangsports unterschieden werden. Die Interaktionsschnittstellen von Konnektoren werden als Rollen bezeichnet und stellen das Gegenstück zu Ports dar. Ebenso wie bei Ports können entsprechend des Datenflusses Sender- und Empfängerrollen unterschieden werden. Die Verknüpfung zwischen Komponenten und Konnektoren erfolgt durch die Assoziation „Verbindung“ zwischen je einem Port und einer Rolle. Gemäß der Richtung des Datenflusses kann entweder eine Senderrolle mit einem Ausgangsport oder eine Empfängerrolle mit einem Eingangsport assoziiert werden. Eine direkte Verbindung zwischen zwei Komponenten bzw. Konnektoren ist nicht zulässig. Die grafische Notation definiert dafür eine verkürzte Schreibweise, die eine vereinfachte Modellierung ermöglicht (siehe Anhang B).

Ein gemeinsames Konzept von Komponenten und Konnektoren ist die Parametrisierbarkeit. Dies wird durch die gemeinsame Oberklasse „ParametrisierbaresElement“ ausgedrückt. Beide Elemente können keinen bzw. beliebig viele Parameter besitzen. Diese stellen die Anknüpfungspunkte zur Steuerung der Adaption durch Kontextinformationen dar. Im Folgenden sollen die Elemente zur Beschreibung der Architektur von Adaptionsgraphen näher erläutert werden.

4.3.2.1 Parametrisierbares Element

Die Klasse „ParametrisierbaresElement“ repräsentiert im Meta-Modell Elemente, die keinen aber auch beliebig viele Parameter enthalten können.

4.3.2.2 Parameter

Das Element „Parameter“ stellt eine Unterklasse der Klasse „GetypterWert“ dar. Es erbt damit die Attribute von „GetypterWert“. Jeder Parameter wird im Meta-Modell genau einem parametrisierbaren Element zugeordnet. Die grafische Notation eines Parameters ist in Abbildung 4-6 dargestellt. Ein Parameter kann in der Form eines getypten Wertes wie folgt notiert werden:

Parametername=(Typ, Einheit, Standardwert, Einschränkung).

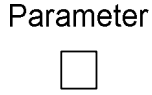


Abbildung 4-6: Grafische Notation eines Parameters

4.3.2.3 Signatur

Eine Signatur beschreibt auf der Modellebene (M1) den Typ von Ports und Rollen. Im einfachsten Fall können Ports bzw. Rollen einen bestimmten Objekttyp empfangen bzw. versenden. Da im Modell jedoch Datenobjekte zusammengefügt bzw. zerlegt, und Ströme von Datenobjekten vereinigt bzw. getrennt werden können, können Signaturen nicht immer durch genau einen Objekttyp beschrieben werden, sondern definieren zum Teil eine Menge von Objekttypen. Eine Signatur wird deshalb generell als Menge von Objekttypen definiert. Eine Signatur, die einen einfachen Objekttyp beschreibt, wird als Menge mit einem Element definiert. Mehrere Elemente einer Menge bezeichnen alternative Objekttypen.

Im Meta-Modell wird eine Signatur auf der Basis des in OCL [OMG03c] definierten Datentyps Set definiert. Dieser ist ein Template-Typ mit einem Parameter T. Eine Signatur wird durch die Ersetzung des Parameters T durch den Typ „Objekttyp“ festgelegt. Dies wird gemäß OCL durch Set(Objekttyp) notiert. Auf der Ebene des Modells ist eine Signatur damit jede Menge von konkreten Objekttypen. Eine leere Menge repräsentiert einen „ungebundenen“ Port bzw. eine „ungebundene“ Rolle. Ungebundene Signaturen werden erst durch die Verknüpfung in Adaptionsgraphen festgelegt.

Eine Signatur kann entsprechend der OCL-Notation durch einen Ausdruck Set{$Objekttyp_1$, $Objekttyp_2$, …, $Objekttyp_n$} erzeugt werden. In OCL werden weiterhin Operationen für Sets definiert. Deren Semantik entspricht der der mathematischen Mengenoperationen. Im Rahmen der Arbeit wird deshalb aus Gründen der einfacheren Lesbarkeit die mathematische Notation für Mengenoperationen verwendet.

4.3.2.4 Ports

Ports repräsentieren die Schnittstellen einer Komponente zur Interaktion mit ihrer Umgebung. Für die Modellierung von Adaptionsgraphen unterliegt jeder Port der Einschränkung, dass er Datenobjekte entsprechend genau einer Signaturdefinition entweder empfangen oder versenden kann. Die Richtung des Datenflusses wird durch die beiden Unterklassen von „Port“, „Eingang“ und „Ausgang“ repräsentiert. Diese Einschränkung erlaubt es,

von den konkreten Operationen des Ports im Sinne einer Schnittstelle zu abstrahieren. Eine Komponente empfängt über einem Eingangsport Datenobjekte entsprechend der definierten Signatur bzw. kann Datenobjekte entsprechend der Signaturdefinitionen der Ausgangsports versenden. Ein Port wird im Modell durch einen innerhalb des Namensraumes der Komponente eindeutigen Namen, eine Signatur und eine Richtung (Eingangs- oder Ausgangsport) definiert. Ein Port wird genau einer Komponentendefinition zugeordnet. Von den Spezifika der Datenübertragung wird durch diese Portdefinition abstrahiert. Art und Protokoll von Interaktionen werden separat und unabhängig von Komponenten in Konnektoren beschrieben. Konkrete Operationen eines Ports können auf plattformabhängiger Ebene hinzugefügt werden. Ein- und Ausgangsports werden entsprechend Abbildung 4-7 notiert. Ein Port wird wie folgt definiert:

```
Portname=(Typ, Signatur).
```

„Typ“ bezeichnet den Typ des Ports. Dieser kann die Werte „Eingang“ und „Ausgang“ annehmen. Die Signatur eines Ports kann einen bzw. mehrere Objekttypen definieren oder ungebunden sein.

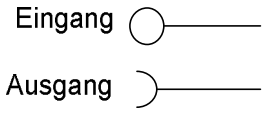


Abbildung 4-7: Gafische Notation von Ein- und Ausgangsports

4.3.2.5 Komponenten

Komponenten repräsentieren in Architekturbeschreibungssprachen die primären Elemente der Verarbeitung. Das Attribut „istAktiv“ definiert die Aktivität einer Komponente. Aktive Komponenten besitzen einen eigenen Kontrollfluss (Thread), passive Komponenten werden dagegen erst durch das Eintreffen von Datenobjekten aktiviert. Die einzelnen Komponententypen sowie Details zur Aktivierung werden in den Abschnitten 4.4.2 und 4.4.5 erläutert.

In der Modellierung werden sie zur Beschreibung des überwiegenden Teils der Mechanismen zur Parameteradaption sowie zur Abbildung einiger Mechanismen zur Strukturadaption entsprechend der Klassifikation aus Kapitel 3 verwendet. Eine Komponente repräsentiert im Modell eine Verarbeitungsoperation, mit der alle Elemente einer Komponente, d. h. Ports und Parameter, in Zusammenhang stehen. Komponenten kapseln damit genau einen Adaptionsmechanismus, wodurch sich die Transparenz und insbesondere auch die Wiederverwendbarkeit von Mechanismen erhöht.

Ein wesentliches Merkmal der Basismechanismen ist die Parametrisierbarkeit, über die eine Steuerung der Adaptionsverarbeitung innerhalb der Komponenten möglich wird. Im Meta-Modell wird dieses Merkmal explizit in Form von Parametern beschrieben, die Komponenten zugeordnet werden. Komponenten sind deshalb im Meta-Modell Unterklassen der Klasse „ParametrisierbaresElement“. Die Verarbeitungsoperation kann zur Laufzeit auf die Werte dieser Parameter zugreifen. Das Element „Parameter“ des Meta-Modells dient der expliziten Beschreibung dieser Parameter und ermöglicht die Definition von Parametern außerhalb der Komponente. Eine Komponente kann keinen oder auch beliebig viele Parameter besitzen. Jeder Parameter besitzt einen innerhalb des Namensraumes der Komponente eindeutigen Namen.

Komponenten besitzen außerdem Schnittstellen, über die sie mit anderen Komponenten interagieren können. Diese werden durch Ports modelliert, die Datenobjekte entweder ein- oder ausgeben. Welche Datenobjekte ein- bzw. ausgegeben werden können, wird durch die Typsignatur festgelegt, die jeder Port definiert. Die graphische Notation einer Komponente sowie deren Ports und Parameter wird in Abbildung 4-8 dargestellt. Eingangsports werden in Anlehnung an die Notation von Schnittstellen in UML 2.0 [OMG03a] als angebotene, d. h. von der Komponente implementierte Schnittstellen und Ausgangsports als geforderte, d. h. von der Komponente benutzte Schnittstellen dargestellt.

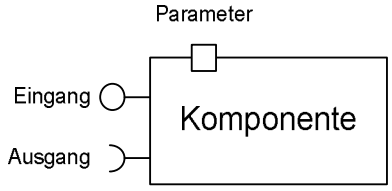


Abbildung 4-8: Grafische Notation einer Komponente

4.3.2.6 Rollen

Rollen repräsentieren die Interaktionsschnittstelle eines Konnektors. Ebenso wie Ports dienen Rollen zum Austausch von Datenobjekten. In Korrespondenz mit Eingangs- und Ausgangsports können Sender- und Empfängerrollen unterschieden werden, die die Richtung des Datenflusses repräsentieren. Ebenso wie Ports werden Rollen durch einen Namen und eine Signatur definiert. Der Name einer Rolle muss innerhalb des Namensraumes des zugehörigen Konnektors eindeutig sein. Jede Rolle ist genau einem Konnektor zugeordnet. Eine Rolle wird wie folgt definiert:

```
Rollenname=(Typ, Signatur).
```

„Typ“ bezeichnet den Typ der Rolle. Dieser kann die Werte „Sender“ und „Empfänger“ annehmen. Die Signatur einer Rolle wird initial als ungebunden festgelegt. Dies entspricht der Semantik eines Konnektors, der Datenobjekte unabhängig von deren Objekttyp übertragen kann. Die Definition der Signaturen von Rollen erfolgt durch die Verbindung mit Ports in Adaptionsgraphen (siehe Abschnitt 4.4.4). Rollen können entsprechend Abbildung 4-9 notiert werden.

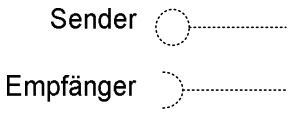


Abbildung 4-9: Grafische Notation von Sender- und Empfängerrollen

4.3.2.7 Konnektoren

Konnektoren kapseln in Architekturbeschreibungssprachen Aspekte der Kommunikation zwischen Komponenten. Konnektoren dienen in Meta-Modell der Vermittlung von Daten. Konnektoren enthalten Kommunikationsprotokolle sowie Mechanismen zur Verzweigung (alternative Pfade), Parallelisierung (parallele Pfade) und der Zusammenführung mehrerer Verbindungen (Sequentialisierung). Sie erhalten Datenobjekte über Ausgangsports sendender Komponenten und vermitteln diese an Eingangsports empfangender Komponenten. Insbesondere gilt, dass ein Konnektor alle erhaltenen Datenobjekte auch vermittelt. Dies

setzt zunächst eine zuverlässige Kommunikation voraus und würde unzuverlässige Kommunikationsprotokolle wir UDP ausschließen. Eine Beschreibung unzuverlässiger Protokolle bzw. von Verlusten von Datenobjekten, welche die oben genannte Bedingung erfüllt, ist in Abschnitt 4.4.3.6 zu finden.

Konnektoren definieren Schnittstellen in Form von Rollen und enthalten zum Teil Verarbeitungslogik, die ebenfalls parametrisierbar sein kann (Paketgröße, Timeout, etc.). Konnektoren sind deshalb ebenso wie Komponenten eine Unterklasse der Klasse „ParametrisierbaresElement“ und können keinen aber auch mehrere Parameter enthalten. Ein Konnektor besitzt je nach Konnektortyp (siehe Abschnitt 4.4.3) mindestens eine Sender- und eine Empfängerrolle. Der Objekttyp einer Rolle ist zunächst „ungebunden“. und wird erst durch die Verbindung mit einem Port definiert. Die graphische Notation eines Konnektors ist in Abbildung 4-10 dargestellt.

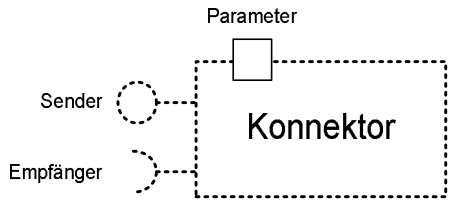


Abbildung 4-10: Grafische Notation eines Konnektors

4.3.2.8 Verbindung

Das Element „Verbindung“ modelliert die Verknüpfung einer Rolle mit einem Port. Es wird jeweils genau eine Rolle an genau einen Port gebunden. Eine Verbindung kann nur zwischen einer Senderrolle und einem Ausgangsport bzw. zwischen einer Empfängerrolle und einem Eingangsport erfolgen. Die grafische Notation von Verbindungen zwischen Rollen und Ports ist in Abbildung 4-11 dargestellt.

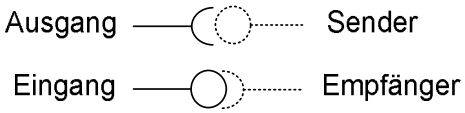


Abbildung 4-11: Grafische Notation von Verbindungen zwischen Rollen und Ports

4.3.2.9 Adaptionsgraph

Adaptive Anwendungen werden durch das Meta-Modell in Form von Adaptionsgraphen beschrieben. Diese repräsentieren adaptive Anwendungen aus der Sicht der Adaption. Durch die Kombination mit weiteren Sichten können vollständige Anwendungen modelliert werden. Die Basis der Beschreibung bilden die Elemente „Komponente“ und „Konnektor“, durch die die Struktur adaptiver Anwendungen beschrieben werden kann. Diese Elemente werden durch die Verknüpfung von Rollen und Ports zu Adaptionsgraphen kombiniert. Ein Adaptionsgraph besteht aus mindestens zwei Komponenten (einer Datenquelle und einer Datensenke) sowie einem Konnektor. Innerhalb des Graphen müssen alle Rollen und Ports sowie Parameter gebunden sein. Abbildung 4-12 stellt einen minimalen Adaptionsgraphen dar. Die weiteren Elemente zur Beschreibung von Adaptionsgraphen werden in den folgenden Abschnitten schrittweise eingeführt.

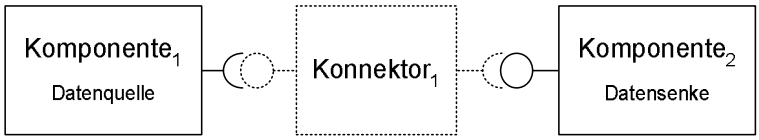


Abbildung 4-12: Die Notation eines minimalen Adaptionsgraphen

4.3.3 Die Modellierung von Meta-Informationen zur Steuerung der Adaption

Meta-Informationen ermöglichen neben Kontext die Steuerung der Adaption innerhalb eines Adaptionsgraphen. Annotationen werden innerhalb einer Komponente geschrieben, d. h zu einem Datencontainer hinzugefügt, und können von nachfolgend ausgeführten Komponenten gelesen werden. Wann dies während der Verarbeitung einer Komponente erfolgt, wird durch die Definition von Verarbeitungspunkten festgelegt. Diese Elemente werden an Ports gebunden und repräsentieren analog dem Konzept der Interceptoren [OMG01] Punkte zum Einfügen von Verarbeitungslogik in Komponenten. An den definierten Verarbeitungspunkten besteht Zugriff auf die am Port eintreffenden bzw. zu versendenden Datenobjekte. An diesen kann damit sowohl das Schreiben als auch das Lesen von Annotationen erfolgen. Außerdem können an diesen Punkten Informationen über die Datenobjekte gelesen und in Form von Meta-Informationen explizit zur Verfügung gestellt werden. Die sind z. B. die Größe des Datenobjektes in Byte oder die Farbtiefe und die räumlichen Maße eines Bildes. Die Elemente zur expliziten Beschreibung von Meta-Informationen und der Zugriff von Komponenten auf diese können durch die Elemente in Abbildung 4-13 beschrieben werden.

4.3.3.1 Annotation

Eine Annotation wird, wie in Abschnitt 4.3.1.3 beschrieben, durch eine Menge von getypten Werten sowie einen eindeutigen Identifikator definiert. Eine Annotation steht mit mehreren Komponenten in Beziehung. Diese können lesend bzw. schreibend auf Annotationen zugreifen.

4.3.3.2 Vorverarbeitung

Jedes Element dieser Klasse wird genau einem Eingangsport zugewiesen. Damit definiert das Element „Vorverarbeitung“ einen Verarbeitungspunkt, an dem Verarbeitungslogik eingefügt werden kann, die nach dem Empfang eines Datenobjektes an diesem Port und vor der Abarbeitung der Komponentenlogik ausgeführt wird (siehe Abschnitt 4.4.5). Die grafische Notation von Verarbeitungspunkten ist in Abbildung 4-14 dargestellt.

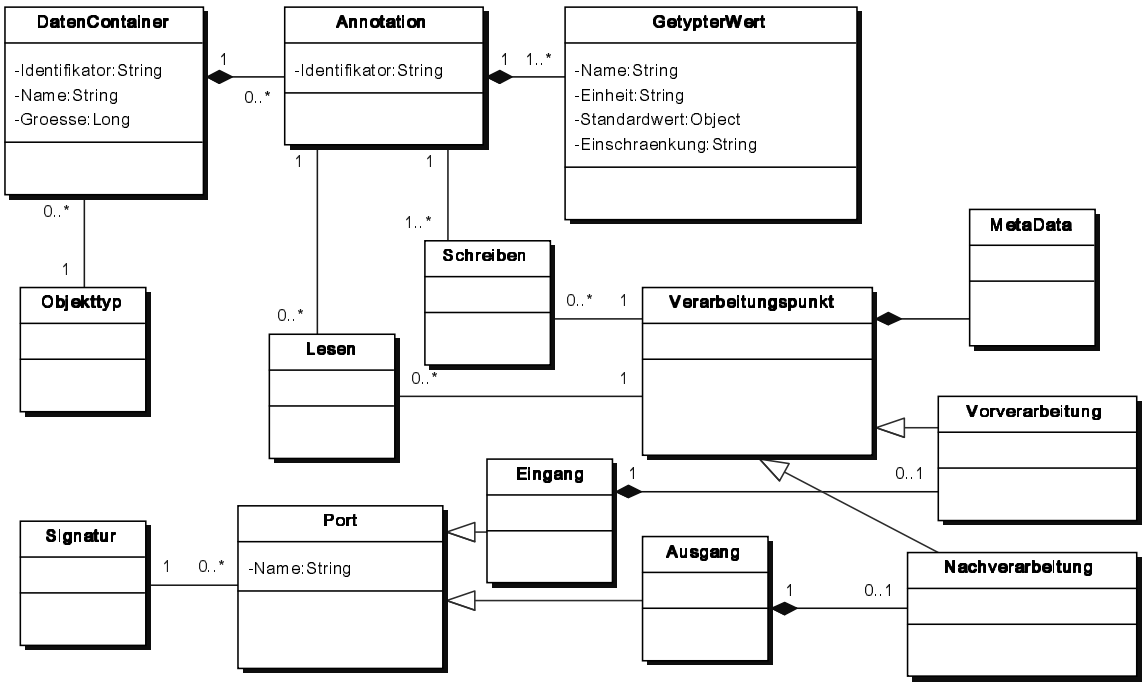


Abbildung 4-13: Das Klassen-Diagramm der Elemente zur Definition von Annotationen

4.3.3.3 Nachverarbeitung

Jedes Element dieser Klasse wird genau einem Ausgangsport zugewiesen. Damit definiert das Element „Nachverarbeitung“ einen Verarbeitungspunkt, an dem Verarbeitungslogik eingefügt werden kann, die nach der Abarbeitung der Komponentenlogik und vor dem Versenden eines Datenobjektes über diesen Port ausgeführt wird. Die grafische Notation von Verarbeitungspunkten ist in Abbildung 4-14 dargestellt.

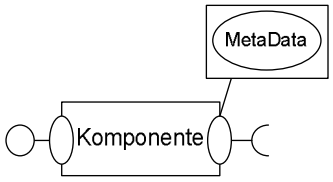


Abbildung 4-14: Grafische Notation von Vor- und Nachverarbeitungspunkten mit und ohne Metadaten

4.3.3.4 Lesen

Das Element „Lesen“ definiert eine Beziehung zwischen einem Verarbeitungspunkt und einer Annotation. Diese beschreibt, dass an dem Verarbeitungspunkt die Meta-Informationen jedes Datenobjektes gelesen werden, das den Verarbeitungspunkt durchläuft. Die grafische Notation einer „Lesen“-Beziehung ist in Abbildung 4-15 dargestellt.

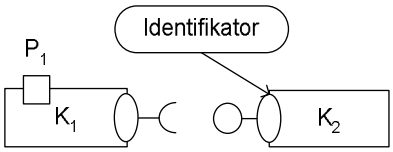


Abbildung 4-15: Grafische Notation einer Lesen-Beziehung

4.3.3.5 Schreiben

Das Element „Schreiben“ definiert eine Beziehung zwischen einem Verarbeitungspunkt und einer Annotation. Diese beschreibt, dass an dem Verarbeitungspunkt Meta-Informationen zu jedem Datenobjekt hinzugefügt werden, das den Verarbeitungspunkt durchläuft. Die grafische Notation einer „Schreiben“-Beziehung ist in Abbildung 4-16 dargestellt.

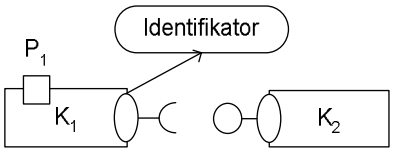


Abbildung 4-16: Grafische Notation einer Schreiben-Beziehung

4.3.3.6 Meta-Daten

Das Element „MetaData“ repräsentiert Meta-Informationen, die an einem Verarbeitungspunkt zur Verfügung stehen. Dies können Informationen über Datenobjekte sein, die den Verarbeitungspunkt durchlaufen aber auch komponenteninterne Informationen. Jedes Element dieser Klasse ist genau einem Verarbeitungspunkt zugeordnet. Die grafische Notation sowie die Textnotation werden vom Element „KontextWert“ geerbt, das wiederum von „GetypterWert“ abgeleitet wurde (siehe Abbildung 4-17 und Abschnitt 4.3.4). Die Notation von Meta-Daten in Textform erfolgt folgendermaßen:

```
MetaData=(Bezeichner, Typ, Einheit, Standardwert, Einschränkung).
```

Abbildung 4-17: Grafische Notation von Kontextwerten

4.3.3.7 Verarbeitungspunkt

Der Zugriff von Komponenten auf Annotationen wird in Form von Verarbeitungspunkten definiert. An einem Verarbeitungspunkt kann auf mehrere Annotationen sowohl lesend als auch schreibend zugegriffen werden. Außerdem können an einem Verarbeitungspunkt beliebig viele Informationen in Form von Meta-Daten bereitgestellt werden. Die Varianten der graphischen Notation von Verarbeitungspunkten enthält Abbildung 4-18. In dieser werden ein Vorverarbeitungspunkt ohne Meta-Daten sowie ein Nachverarbeitungspunkt dargestellt, an dem Metadaten definiert werden.

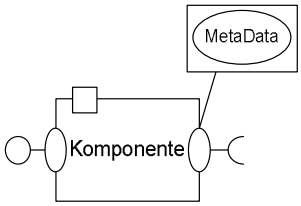


Abbildung 4-18: Grafische Notation eines Verarbeitungspunktes mit und ohne Meta-Daten

4.3.4 Steuerung der Adaption durch Kontext

Kontext repräsentiert Informationen über die Ausführungsumgebung von Anwendungen und liefert damit wichtige Informationen, die zur Steuerung sowohl der Parameter- als auch der Strukturadaption verwendet werden können. Dabei können entsprechend der Anforderungen aus Kapitel 1 sowohl Informationen über den Zustand (Endgerät, Netzwerk, Benutzeranforderungen und Anwendungssituation) als auch die dynamische Änderung des Zustands der Ausführungsumgebung genutzt werden. In Abschnitt 2.4 werden Ansätze zur Realisierung eines Kontextdienstes beschrieben. Dieser hat die Aufgabe, Kontextinformationen aus Roh- und Messdaten von Sensoren aufzubereiten und für eine Nutzung in Anwendungen zur Verfügung zu stellen. Informationen wie die verfügbare Datenrate oder die Größe des Displays des Endgerätes repräsentieren unabhängige Informationen über die Ausführungsumgebung und müssen innerhalb einer Anwendung auf spezifische Steuerinformationen für die Adaption abgebildet werden. Beispielsweise muss die Information über die Displaygröße des Endgerätes auf Parameter zur Skalierung und Veränderung der Farbtiefe abgebildet werden, um ein Bild entsprechend anzupassen. Ebenso müssen sich dynamisch ändernde Informationen innerhalb der Anwendung verarbeitet werden können, um in Reaktion auf Änderungen spezifische Aktionen auslösen.

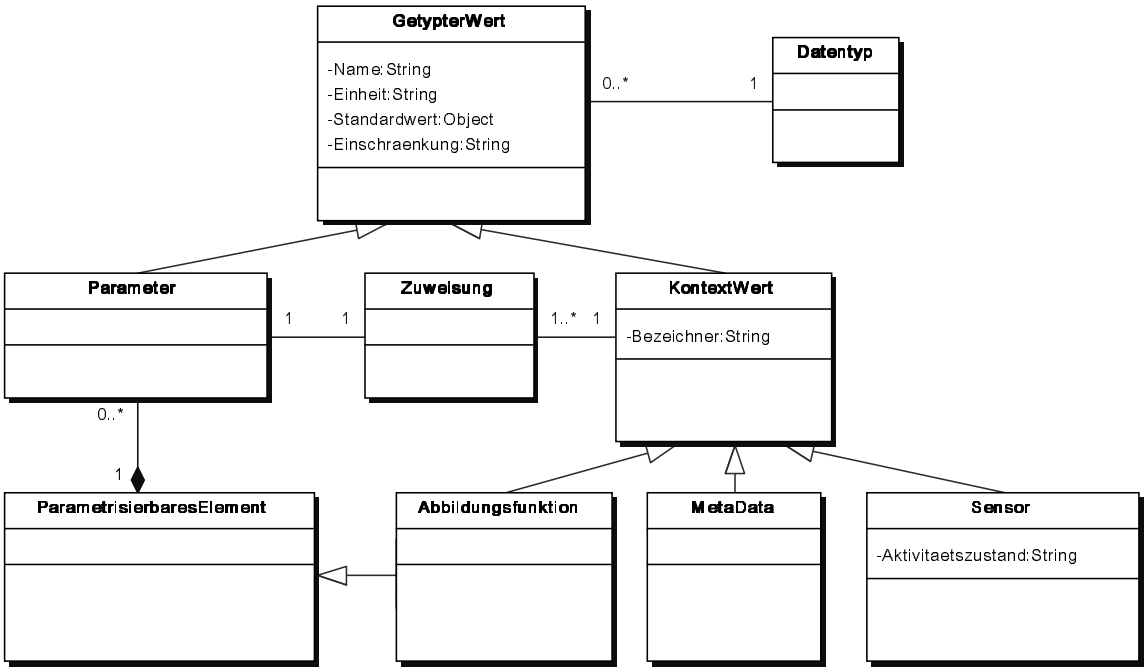


Abbildung 19: Das Klassendiagramm der Elemente zur Abbildung von Kontext auf Parameter

Die Abbildung von Konext auf Parameterwerte soll gemäß der Anforderung A4.3 explizit beschrieben werden können. Dazu dienen die Elemente „Sensor“, „Abbildungsfunktion“,

„Kontextwert“ und „Zuweisung“ (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**). Diese ermöglichen die Definition von Kontextwerten sowie deren Abbildung auf Parameter.

4.3.4.1 KontextWert

Das Element „KontextWert“ definiert die Oberklasse aller Elemente, die auf Parameter abgebildet werden können. Kontextwerte sind eine Unterklasse der Klasse „Getypter-Wert“. Das Element „KontextWert“ definiert zusätzlich das Attribut „Bezeichner“, das eine Beschreibung des Kontextes enthält. Die Unterklassen von „KontextWert“ „Sensor“ und „MetaData“ werden entsprechend Abbildung 4-20 dargestellt. Abbildungsfunktionen enthalten zusätzlich Parameter (siehe 4.3.4.4). Die Notation eines Kontextwertes in Textform erfolgt folgendermaßen:

```
Kontextname=(Bezeichner, Typ, Einheit, Standardwert, Einschränkung).
```

Abbildung 4-20: Grafische Notation von Kontextwerten

4.3.4.2 Zuweisung

Die Abbildung eines Kontextwertes auf einen Parameter wird durch das Element „Zuweisung“ definiert. Ein Kontextwert kann mehreren Parametern zugewiesen werden, ein Parameter wird jedoch mit genau einem Kontextwert assoziiert. Die grafische Notation einer „Zuweisung“ wird in Abbildung 4-21 dargestellt.

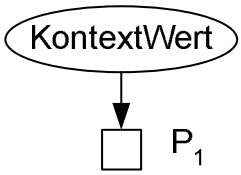


Abbildung 4-21: Grafische Notation einer Zuweisen-Beziehung

4.3.4.3 Sensoren

Sensoren modellieren den Zugriff auf einen Kontextdienst zur Ermittlung genau eines Kontextwertes. Entsprechend der Annahme in Abschnitt 4.2.1 wird ein Kontextdienst vorausgesetzt, der alle geforderten Kontextwerte liefern kann. Sensoren abstrahieren von konkreten Zugriffen sowie der Zugriffsart auf einen Kontextdienst. Die Klasse „Sensor“ ist eine Unterklasse der Klasse „KontextWert“. Sie enthält eine eindeutige Bezeichnung des Kontextwertes, der durch den Sensor repräsentiert wird. Zusätzlich kann für einen Sensor ein Aktivitätszustand festgelegt werden. Mögliche Zustände sind „aktiv“, „passiv“ und „konstant“. Eine detaillierte Erläuterung der Semantik der Zustände enthält Abschnitt 4.4.6. Ein Sensor kann in Textform wie ein getypter Wert, erweitert um den Aktivitätszustand, definiert werden:

```
Sensorname=(Bezeichner, Typ, Einheit, Standardwert, Einschränkung, Aktivitätszustand).
```

Ein Beispiel für einen Sensor ist die horizontale Anzahl der Bildpunkte des Displays eines Endgerätes, die wie folgt notiert werden kann:

S_1=(Device.Display.x, Integer, “pixel”, 1280, “>0”, passiv).

In ähnlicher Weise können Informationen bezüglich des Netzwerkes, der Benutzeranforderungen und der Anwendungssituation modelliert werden. Die graphische Notation wird vom Element “GetypterWert” abgeleitet (siehe Abbildung 4-22). Der Aktivitätszustand eines Sensors wird durch die Stereotypen <<aktiv>>, <<passiv>> und <<konstant>> gekennzeichnet.

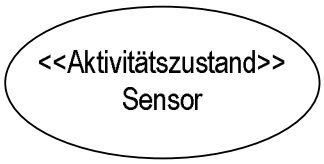


Abbildung 4-22: Grafische Notation eines Sensors

4.3.4.4 Abbildungsfunktion

Abbildungsfunktionen beschreiben den Zugriff auf Kontext und dienen zur Ermittlung von Komponentenparametern aus Informationen über den aktuellen Zustand der Ausführungsumgebung. Das Element „Abbildungsfunktion“ ist sowohl eine Unterklasse der Klasse „KontextWert“ als auch der Klasse „ParametrisierbaresElement“. Es enthält mindestens einen Parameter. Parameterwerte werden durch definierte Funktionen auf einen Resultatwert abgebildet, der wiederum einem Parameter zugewiesen werden kann. Dies ermöglicht die Verwendung von Resultatwerten von Abbildungsfunktionen als Eingabewerte weiterer Abbildungsfunktionen. Damit wird eine schrittweise Berechnung von Parametern von Komponenten und Konnektoren aus Kontextwerten unterstützt. Insbesondere können auf diese Weise Standardfunktionen kombiniert und damit auch auf dieser Ebene eine Wiederverwendung ermöglicht werden. Beispiele für Abbildungsfunktionen sind die Ermittlung des Minimums bzw. Maximums aus mehreren Eingabewerten, Schwellwertfunktionen sowie Berechnungsfunktionen (beispielsweise zur Ermittlung eines Skalierungsfaktors für Bilddaten auf Basis einer gegebenen Datenrate). Das Element „Abbildungsfunktion“ besteht aus der Deklaration von Parametern, der Definition des Ergebniswertes sowie der Beschreibung der Funktion. Der Ergebniswert entspricht einem Kontextwert. Die grafische Notation von Abbildungsfunktion erweitert die von Kontextwerten um Parameter (siehe Abbildung 4-23).

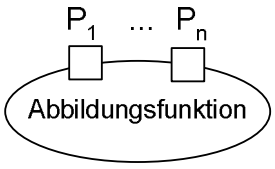


Abbildung 4-23: Grafische Notation von Abbildungsfunktionen

4.3.5 Zusammengesetzte Komponenten und Konnektoren

Das Konzept mehrerer Hierarchieebenen für die Architekturbeschreibung erlaubt ein Zusammenfassen bzw. Zerlegen von Architekturen in eine abstraktere bzw. detaillierte Darstellung. Für die Modellierung soll die Komposition zusammengesetzter Konnektoren und

Komponenten ermöglicht werden. Damit können komplexe Mechanismen aus Basismechanismen komponiert werden. Ebenso können Mechanismen zerlegt und damit detailliertere beschrieben werden. Die zur Modellierung zusammengesetzter Komponenten und Konnektoren definierten Elemente enthält Abbildung 4-24.

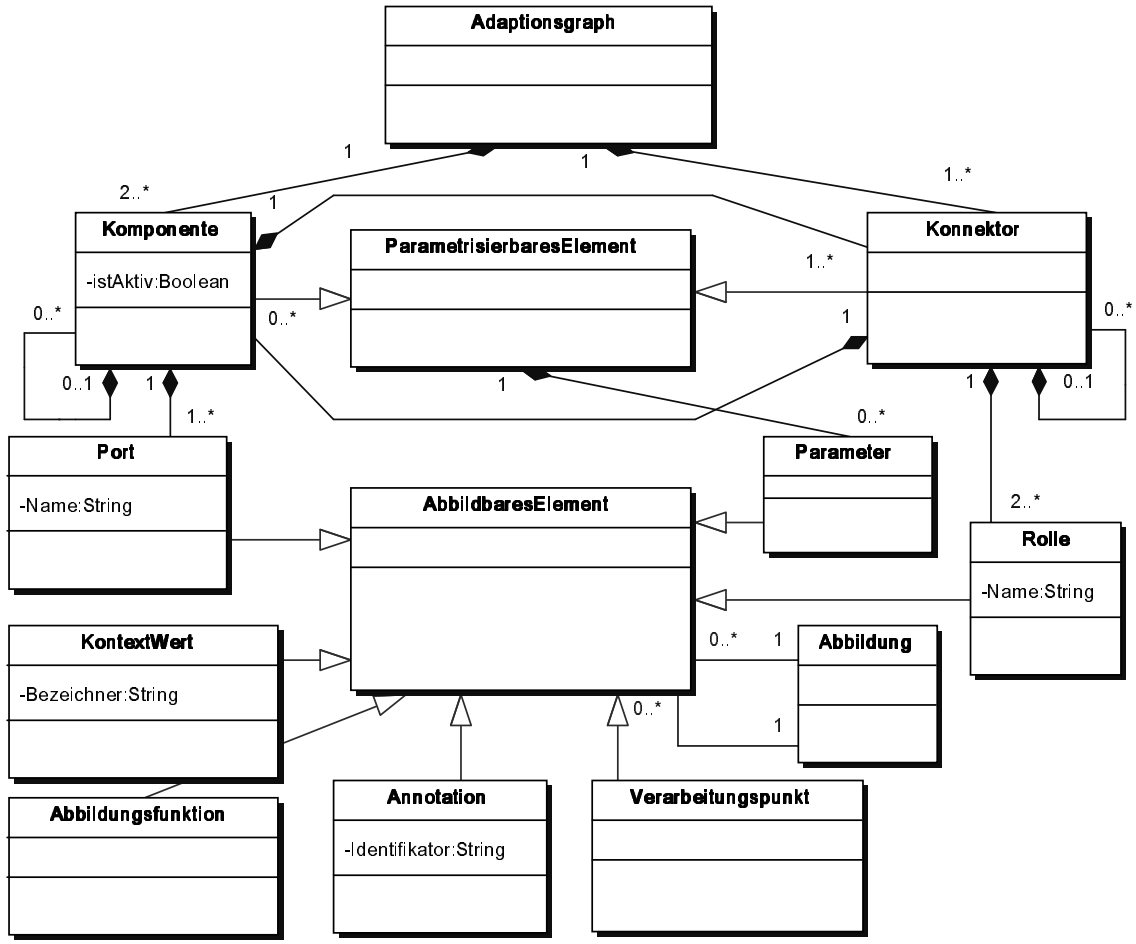


Abbildung 4-24: Die Elemente zur Beschreibung zusammengesetzter Komponenten und Konnektoren

Komponenten und Konnektoren können dementsprechend weitere Komponenten und Konnektoren enthalten, die die Struktur der Elemente auf der nächst tieferen Hierarchieebene beschreiben. Außerdem besitzen die zusammengesetzten Komponenten und Konnektoren Parameter und Ports bzw. Rollen, die durch entsprechende Elemente der internen Struktur auf die nächst höhere Ebene abgebildet werden. Ebenso sind Annotationen, Abbildungsfunktionen und Sensoren Bestandteile von Komponenten und Konnektoren. Diese Elemente können ebenfalls auf die nächst höhere Hierarchieebene abgebildet werden. Dies entspricht dem Konzept der „Representation Maps“ in ACME (siehe Abschnitt 2.5.4) bzw. Dependencies in UML 2.0 [OMG03a].

Die Elemente „Abbildungsfunktion“, „Annotation“ und „Sensor“ sind ebenfalls Bestandteile von Komponenten und Konnektoren im Sinne einer Kompositionsassoziation. Diese Assoziationen sind aus Gründen der Übersichtlichkeit nicht in Abbildung 4-24 dargestellt.

4.3.5.1 AbbildbaresElement

Die Klasse „AbbildbaresElement“ bezeichnet die Oberklasse von Elementen, die zwischen benachbarten Hierarchieebenen abgebildet werden können. Unterklassen sind „Port“, „Rolle“, „Parameter“, „Annotation“, „KontextWert“ und „Abbildungsfunktion“.

4.3.5.2 Abbildung

Das Element „Abbildung“ repräsentiert eine Beziehung zwischen gleichartigen Elementen zweier Hierarchieebenen. Es beschreibt die Abbildung eines abbildbaren Elementes (Klasse „AbbildbaresElement“) der Hierarchieebene n auf ein abbildbares Element der Hierarchieebene n+1. Es gilt die Beschränkung, dass ein abbildbares Element nur auf ein Element gleichen Typs abgebildet werden kann. Demnach kann ein Eingangsport der Hierarchieebene n nur auf einen Eingangsport der Hierarchieebene n+1 abgebildet werden. Gleiches gilt für Ausgangsports, Sender- und Empfängerrollen, Parameter, Annotationen, Abbildungsfunktionen und Kontextwerte. Für Ports und Rollen muss insbesondere die Signatur übereinstimmten. Bei Parametern, Kontextwerten, Abbildungsfunktionen und Annotationen müssen alle Attribute bzw. Bestandteile sowie deren Typen übereinstimmen. Die grafische Notation der Abbildungsbeziehung ist in Abbildung 4-25 dargestellt.

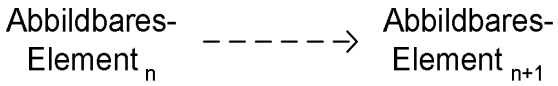


Abbildung 4-25: Grafische Notation einer Abbildungsbeziehung

4.3.5.3 Zusammengesetzte Komponenten

Eine zusammengesetzte Komponente beschreibt die Kombination mehrerer Komponenten und Konnektoren. Das Ergebnis der Kombination ist wiederum eine Komponente, die somit der Definition einer Komponente im Meta-Modell genügen muss. Zusammengesetzte Komponenten besitzen also ebenso wie Komponenten Parameter und Ports. Die Definition dieser Elemente erfolgt durch die Abbildung von Elementen der enthaltenen „inneren“ Komponenten und Konnektoren auf die Elemente der „äußeren“ Komponente. Die Abbildung von Ports und Parametern erfolgt über das Element „Abbildung“. Der Zusammenhang zwischen zwei benachbarten Abbildungsebenen sowie die Notation einer „Abbildung„ werden in Abbildung 4-26 dargestellt.

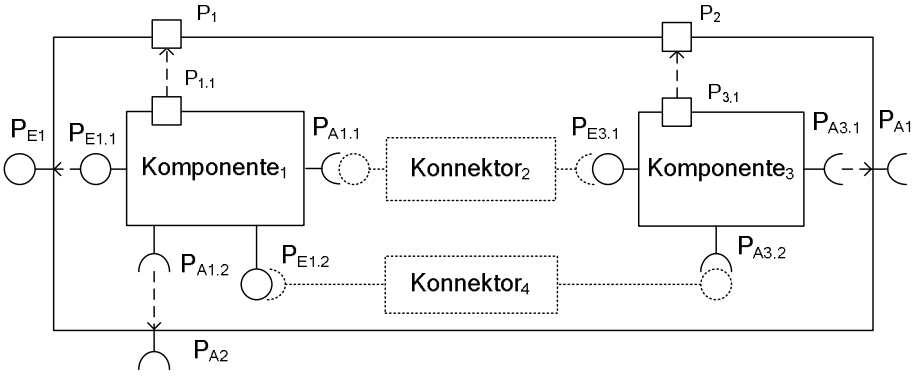


Abbildung 4-26: Definition einer zusammengesetzten Komponente

4.3.5.4 Zusammengesetzte Konnektoren

Analog zu zusammengesetzten Komponenten können durch die Kombination von Konnektoren und Komponenten zusammengesetzte Konnektoren definiert werden. Diese besitzen ebenso wie einfache Konnektoren des Meta-Modells Rollen und Parameter. Die Definition dieser Elemente erfolgt durch die Abbildung von Elementen der enthaltenen Komponenten und Konnektoren auf die Elemente des „äußeren" Konnektors. Die Abbildung von Rollen und Parametern erfolgt über das Element „Abbildung". Der Zusammenhang zwischen zwei benachbarten Abbildungsebenen wird in Abbildung 4-27 dargestellt.

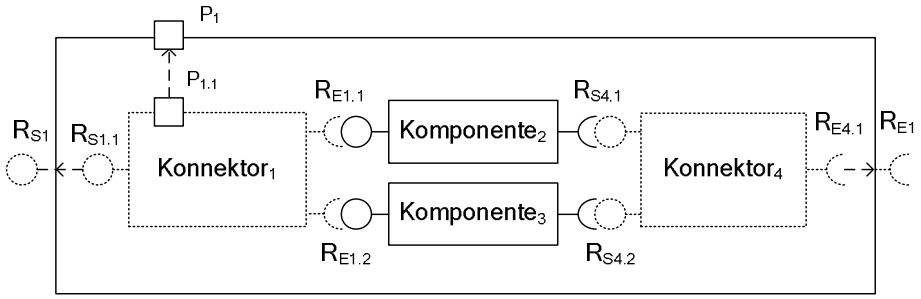


Abbildung 4-27: Definition eines zusammengesetzten Konnektors

4.4 Semantik des Meta-Modells

Nach der formalen Beschreibung der Elemente des Meta-Modells soll nun deren Semantik betrachtet werden. Dabei sollen vor allem die wesentlichen Konzepte sowie die Zusammenhänge zwischen den einzelnen Elementen verdeutlicht werden.

4.4.1 Die Grundstruktur von Adaptionsgraphen

Die Elemente des Meta-Modells ermöglichen die Beschreibung adaptiver Anwendungen in Form gerichteter Graphen. Die Knoten des Graphs werden durch Komponenten und Konnektoren repräsentiert, die Kanten durch die Verknüpfung von Ports und Rollen. Die Komponenten enthalten die Verarbeitungslogik, d. h. die Mechanismen zur Adaption von Anwendungsdaten sowie auf der Ebene der Kommunikation die Mechanismen zur Zwischenspeicherung und zur Zugriffssteuerung. Konnektoren modellieren auf der Ebene der Kommunikation die Mechanismen zur Übertragung. Die Adaption der Anwendungsstruktur wird durch alternative Pfade beschrieben. Die Logik zur Auswertung von Bedingungen für die Auswahl eines Pfades und zur Verteilung von Datenobjekten können Komponenten in spezifischer und Konnektoren in generischer Form enthalten. Die Mechanismen zum Verzweigen, Parallelisieren und Vereinigen von Kommunikationsverbindungen werden durch Konnektoren modelliert. Anwendungsspezifische Entscheidungen zur Auswahl eines Pfades werden dagegen durch Komponenten modelliert. Ein Beispiel dafür ist die Verzweigung durch eine Annotation, die in einer Komponente ausgeführt wird. Nachfolgend sollen Typen von Komponenten und Konnektoren betrachtet werden.

4.4.2 Komponententypen

Komponenten empfangen Datenobjekte, verarbeiten diese und geben Datenobjekte aus. Die Datenobjekte werden außerdem in bestimmten Komponenten erzeugt bzw. explizit aus dem Adaptionsgraph entfernt. Anhand des Verhaltens von Komponenten in Bezug auf die

Verarbeitung, Erzeugung bzw. Terminierung von Datenobjekten können die folgenden vier Komponententypen unterschieden werden:

1. *Datenquellen*: Komponenten dieses Typs erzeugen Anwendungsdaten und versenden diese über mindestens einen Ausgangsport.
2. *Datensenken*: Komponenten dieses Typs entfernen empfangene Datenobjekte explizit aus dem Adaptionsgraphen. Sie besitzen mindestens einen Eingangsport.
3. *Verarbeitungskomponenten:* Komponenten dieses Typs empfangen jeweils ein Datenobjekt, verarbeiten dieses und versenden das Ergebnisobjekt der Verarbeitung ohne Zwischenspeicherung. Verarbeitungskomponenten besitzen mindestens einen Ein- und einen Ausgangsport.
4. *Speicherkomponenten:* Komponenten dieses Typs empfangen Datenobjekte und speichern diese in einem durch die Komponente verwalteten Speicher. Dazu enthalten sie zwei Verarbeitungsschritte. Im ersten Verarbeitungsschritt werden empfangene Datenobjekte gespeichert, im zweiten Verarbeitungsschritt werden Daten im Speicher gesucht und gegebenenfalls aus diesem entfernt und versendet. Eine Speicherkomponente besitzt mindestens einen Eingangs- und einen Ausgangsport.

4.4.3 Konnektortypen

Konnektoren repräsentieren die Verbindungen zwischen Komponenten. Sie modellieren Übertragungsprotokolle sowie die Multiplizität einer Verbindung. Einer Komponente bleibt es damit verborgen, an wie viele Empfänger ein von ihr versendetes Datenobjekt vermittelt wird, ob die Kommunikation lokal oder entfernt erfolgt und welches Kommunikationsprotokoll verwendet wird. Entsprechend der Multiplizität der Verbindungen können fünf Typen von Konnektoren unterschieden werden: Sequenz-, Multiplex-, Demultiplex-, Typsortier- und Multicastkonnektoren. Diese repräsentieren für m Sender und n Empfänger Verbindungen der Multiplizität 1:1, 1:n und m:1. Verbindungen der Muliplizät m:n werden im Rahmen der Arbeit nicht betrachtet, da diese durch die Kombination von 1:n- und m:1- Verbindungen modelliert werden können.

4.4.3.1 Sequenzkonnektor

Ein Sequenzkonnektor beschreibt eine Verbindung zwischen einem Sender und einem Empfänger. Er definiert zwei Rollen, eine Senderrolle und eine Empfängerrolle. Sequenzkonnektoren werden entsprechend Abbildung 4-28 notiert.

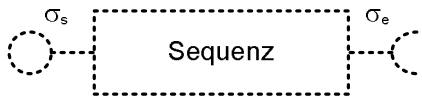


Abbildung 4-28: Sequenzkonnektor für 1:1 Verbindungen

Sequenzkonnektoren stellen eine Oberklasse zur Modellierung von Verbindungen zwischen einem Ausgangs- und einem Eingangsport dar. Durch die Bildung von Unterklassen wie „Lokal“ und „Entfernt“ bzw. durch die Bennenung des Konnektors mit dem entsprechenden Kommunikationsprotokoll kann explizit eine lokale oder entfernte Kommunikation bzw. ein spezifisches Kommunikationsprotokoll beschrieben werden. In Abbildung 4-29 ist eine vereinfachte Notation für einen Konnektor dargestellt. Diese kann im Modell je nach Definition zur Repräsentation von Standardkonnektoren zur lokalen Kommunikati-

on bzw. für nicht näher beschriebene Konnektoren verwendet werden. Konnektoren, die bestimmte Interaktionsmechanismen bzw. Kommunikationsprotokolle repräsentieren, werden entsprechend der Notation von Sequenzkonnektoren in Abbildung 4-28 dargestellt. Die Signaturen der Sender- und Empfängerrolle σ_s bzw. σ_e eines Sequenzkonnektors müssen übereinstimmen, d. h. $\sigma_e=\sigma_s$

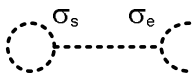


Abbildung 4-29: Vereinfachte Notation eines Sequenzkonnektors

4.4.3.2 Multiplexkonnektor

Multiplexkonnektoren bilden Ströme von Datenobjekten mehrerer paralleler Sender auf eine Sequenz von Datenobjekten ab, die an einen Empfänger vermittelt wird. Konnektoren dieses Typs besitzen mindestens zwei Senderrollen und eine Empfängerrolle. Die Anzahl der Senderrollen ist variabel und kann vom Entwickler in Adaptionsgraphen beliebig definieren werden. Die Signatur der Empfängerrolle σ_e wird durch die Vereinigung der Signaturen aller Senderrollen σ_{si} definiert, d. h. $\sigma_{e=}\sigma_{s1} \cup \sigma_{s2} \cup \ldots \cup \sigma_{sn}$ (mit n = Anzahl der Senderrollen). Dies folgt aus der Forderung, dass Datenobjekte im Adaptionsgraphen explizit terminiert werden müssen. Die Objekttypen aller Sender werden somit beim Empfänger vereinigt. Abbildung 4-30 stellt einen Multiplexkonnektor mit zwei Senderrollen dar.

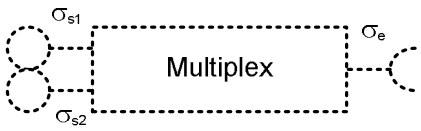


Abbildung 4-30: Multiplexkonnektor für m:1 Verbindungen

4.4.3.3 Demultiplexkonnektor

Demultiplexkonnektoren beschreiben die Vermittlung von Datenobjekten an alternative Empfänger, d. h. jedes Datenobjekt wird an genau einen Empfänger weitergeleitet. Die Auswahl des Empfängers wird im Konnektor getroffen und kann über Parameter durch Kontext gesteuert werden. Ein Demultiplexkonnektor besitzt genau eine Senderrolle und mindestens zwei Empfängerrollen. Die Anzahl der Empfängerrollen ist aber variabel. Der Entwickler kann in Adaptionsgraphen also beliebig viele Empfängerrollen definieren. Für Demultiplexkonnektoren besteht die Forderung, dass die Signaturen aller Empfängerrollen σ_{ei} gleich der Signatur der Senderrolle σ_s sein müssen, d. h. $\sigma_{ei}=\sigma_s$ (mit i=1 bis n und n=Anzahl der Empfängerrollen). Ein Beispiel eines Demultiplexkonnektors mit zwei Empfängerrollen ist in Abbildung 4-31 enthalten.

Abbildung 4-31: Demultiplexkonnektor für 1:n Verbindungen

4.4.3.4 Typsortierkonnektor

Ein Typsortierkonnektor vermittelt Datenobjekte in Abhängigkeit ihres Objekttyps an einen von mehreren alternativen Empfängern. Jedes Datenobjekt wird dabei genau einem Empfänger zugeordnet, Datenobjekte mit gleichem Objekttyp werden dem gleichen Empfänger zugeordnet. Ein Typsortierer besitzt genau eine Senderrolle und mindestens zwei Empfängerrollen. Die Anzahl der Empfängerrollen ist variabel wird aber durch die Anzahl der Objekttypen in der Signatur der Senderrolle begrenzt. Für Typsortierkonnektoren besteht die Forderung, dass die Vereinigung der Signaturen aller Empfängerrollen σ_{ei} gleich der Signatur der Senderrolle σ_s sein müssen, d. h. $\sigma_s=\sigma_{e1} \cup \sigma_{e2} \cup \ldots \cup \sigma_{en}$ (mit n = Anzahl der Empfängerrollen) und die Signaturen der Empfängerrollen paarweise disjunkt sein müssen, d. h. $\sigma_{ei} \cap \sigma_{ej} = \varnothing$ (für alle i, j $\in$ {1..n} und i $\neq$ j und n = Anzahl der Empfängerrollen). Ein Beispiel eines Typsortierkonnektors mit zwei Empfängerrollen enthält Abbildung 4-32.

Abbildung 4-32: Typsortierkonnektor für 1:n Verbindungen

4.4.3.5 Multicastkonnektor

Multicastkonnektoren vermitteln ein Datenobjekt parallel an mehrere Empfänger (UND-Verteilung). Konnektoren dieses Typs besitzen somit eine Senderrolle und mindestens zwei Empfängerrollen. Die Anzahl der Empfängerrollen ist variabel. Der Entwickler kann in Adaptionsgraphen also beliebig viele Empfängerrollen definieren. Die Signatur aller Empfängerrollen σ_{ei} muss gleich der Signatur der Senderrolle σ_s sein, d. h $\sigma_{ei}=\sigma_s$ (mit i = 1 bis n und n = Anzahl der Empfängerrollen). Ein Multicastkonnektor mit zwei Empfängerrollen ist in Abbildung 4-33 dargestellt.

Abbildung 4-33: Multicastkonnektor für 1:n Verbindungen

4.4.3.6 Modellierung unzuverlässiger Verbindungen

Datenobjekte können nur explizit durch eine Datensenke aus einem Adaptionsgraphen entfernt werden. Eine Modellierung von Datenverlusten während der Übertragung kann durch eine Trennung der übertragenen Datenobjekte in erfolgreich übertragene bzw. verlorene Datenobjekte erfolgen. Nach dieser Trennung werden die als verloren markierten Datenobjekte an eine Datensenke vermittelt, welche die Datenobjekte explizit verwirft. Die Auswahl kann durch einen Demultiplexkonnektor und alternativ durch eine Filterkomponente beschrieben werden (siehe Abbildung 4-34).

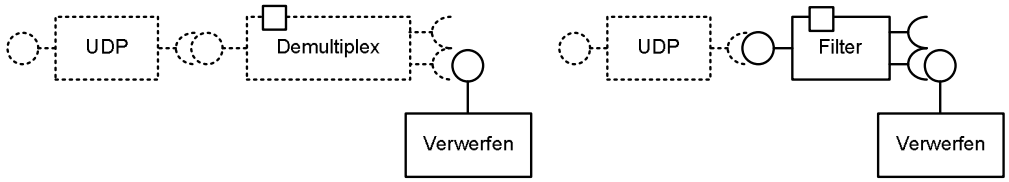


Abbildung 4-34: Alternative Möglichkeiten der Beschreibung einer unzuverlässigen Übertragung am Beispiel eines UDP-Konnektors

Die Parameter des Demultiplexkonnektors bzw. des Filters können außerdem verwendet werden, um den Verlust von Datenobjekten kontextabhängig zu beschreiben. Beispielsweise könnte der Parameter die Verlustwahrscheinlichkeit repräsentieren, die abhängig von der aktuellen Kommunikationsverbindung festgelegt wird.

4.4.4 Typsicherheit im Modell

Die Kombinierbarkeit von Komponenten wird im Modell durch die Typdefinitionen der Ein- und Ausgangsports der Komponenten festgelegt. Es können nur Ports mit kompatiblen Signaturen über Konnektoren miteinander verbunden werden. Bei jeder Verbindung eines Ein- mit einem Ausgangsports muss deshalb eine Überprüfung der Signaturen stattfinden.

Wie in Abschnitt 4.3.2.6 festgelegt, sind die Rollen aller Konnektoren, die nicht Bestandteil eines Adaptionsgraphen sind, ungebunden, d. h. für alle Signaturen σ_i eines Konnektors gilt σ_i = set{}. Konnektoren vermitteln Datenobjekte außerdem ohne Veränderungen des Objekttyps an ihre Empfängerrollen. Weiterhin gilt in diesem Zusammenhang die Festlegung, dass Datenobjekte nur explizit aus dem Graphen entfernt werden können. Alle Datenobjekte, die an einen Konnektor übergeben werden, müssen diesen auch wieder verlassen. Für einen Konnektor mit einer Sender- sowie einer Empfängerrolle entspricht also die Signatur der Senderrolle der der Empfängerrolle. In Abbildung 4-35 kann der Ausgangsport von K_1 mit dem Eingangsport von K_2 über den Konnektor C_1 verbunden werden, wenn für die Signaturen σ_1, σ_2 und σ_3 gilt: $\sigma_1 \subseteq \sigma_3$. Für die Typüberprüfung wird $\sigma_2 = \sigma_1$ gesetzt, da σ_2 ungebunden ist. Dementsprechend muss $\sigma_2 \subseteq \sigma_3$ gelten. Diese Beziehung gilt für Sequenzkonnektoren (siehe Abschnitt 4.4.3). Ähnliches gilt für Multicast- und Demultiplexkonnektoren. Bei diesen entspricht ebenfalls jede Signatur einer Empfängerrolle der Signatur der Senderrolle. Die Typüberprüfung kann für alle Empfängerrollen analog zu den obigen Definitionen erfolgen.

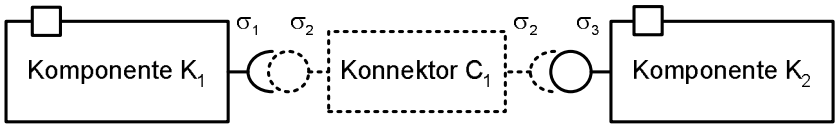


Abbildung 4-35: Typbeziehungen bei der Verbindung zweier Komponenten über einen Sequenzkonnektor

Für Multiplexkonnektoren gelten andere Beziehungen zwischen den Signaturen der Sender- und Empfängerrollen. Der Multiplexkonnektor vereinigt entsprechend seiner Definition an den verschiedenen Senderrollen empfangene Datenobjekte zu einer Sequenz von Datenobjekten an der Empfängerrolle. Dementsprechend gilt für die Signaturen der Rollen des Multiplexkonnektors in Abbildung 4-36 $\sigma_7 = \sigma_4 \cup \sigma_5 \cup \sigma_6$. Komponente K_4 muss demzufolge Datenobjekte aller Signaturen σ_4, σ_5 und σ_6 verarbeiten können, da $\sigma_8 \subseteq \sigma_7$ gelten muss. Allgemein gilt für einen Multiplexkonnektor M mit den Senderrollen S_1 bis S_n

und deren Typen σ_{S1} bis σ_{Sn} sowie der Empfängerrolle E und deren Signatur $\sigma_{E:}$ $\sigma_E = \sigma_{S1} \cup \sigma_{S2} \cup \ldots \cup \sigma_{Sn}$ (mit n = Anzahl der Senderrollen).

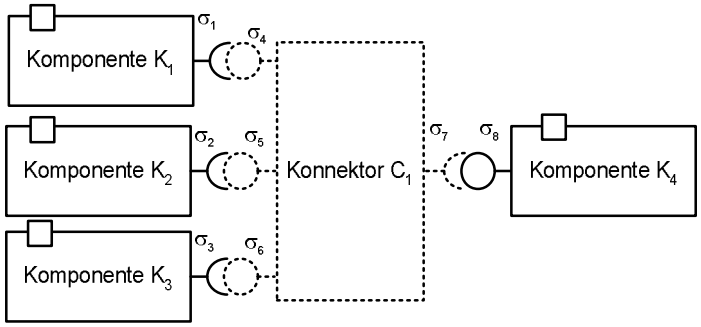


Abbildung 4-36: Typbeziehungen bei der Verbindung von Komponenten über einen Multiplexkonnektor

Ein Typsortierter wählt entsprechend der Signaturen der Datenobjekte eine Empfängerrolle zur Ausgabe aus. Entsprechend der Auswahlbedingung wird jeweils eine Teilmenge der eintreffenden Datenobjekte an die einzelnen Empfängerrollen weitergeleitet. Die Signaturen der Empfängerrollen entsprechen somit ebenfalls Teilmenge der Signatur der Senderrolle. Für Abbildung 4-37 gilt deshalb $\sigma_2 = \sigma_3 \cup \sigma_4$. Die Komponente K_1 muss dementsprechend nach ihrer Verarbeitung Datenobjekte der Signaturen σ_3 und σ_4 ausgeben. Allgemein gilt für einen Typsortierer T mit der Senderolle S_1 und deren Signatur σ_S sowie den Empfängerrollen E_1 bis E_n mit den Signaturen σ_{E1} bis σ_{En}: $\sigma_S = \sigma_{E1} \cup \sigma_{E2} \cup \ldots \cup \sigma_{En}$. Außerdem müssen die Signaturen der Empfängerrollen paarweise disjunkt sein: $\sigma_{Ei} \cap \sigma_{Ej} = \varnothing$ (für alle $i, j \in \{1..n\}$ und $i \neq j$ und n = Anzahl der Empfängerrollen).

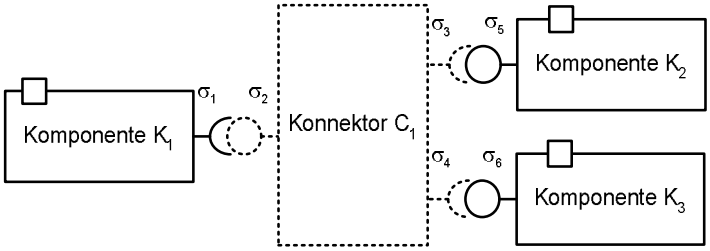


Abbildung 4-37: Typbeziehungen bei der Verbindung zweier Komponenten über einen Konnektor zur typabhängigen Vermittlung von Datenobjekten

4.4.5 Konzepte zur Kommunikation in Adaptionsgraphen

Komponenten können sich in Bezug auf Interaktionen aktiv oder passiv verhalten. Die Beschreibung der Aktivität kann getrennt sowohl für den Empfang als auch das Versenden von Daten betrachtet werden. Ein Sender bzw. Empfänger ist damit entweder aktiv oder passiv. Betrachtet man die Verbindung eines Senders mit einem Empfänger, dann muss einer der beiden Kommunikationspartner aktiv, d. h. Initiator der Datenübertragung [VBT95], und der andere passiv sein. Kommunizieren zwei Objekte (entsprechend der Terminologie der objekt-orientierten Softwareentwicklung) über synchrone Nachrichten, können die beiden Varianten eines aktiven Senders und passiven Empfängers bzw. eines passiven Senders und eines aktiven Empfängers unterschieden werden. Beide Möglichkeiten werden im Sequenzdiagramm in Abbildung 4-38 dargestellt.

Im oberen Teil des Sequenzdiagramms ruft der aktive Sender die Methode sende_Daten(Datenobjekt) des Empfängers auf und übergibt gleichzeitig die Daten. Die Signalisierung erfolgt gekoppelt mit der Übergabe der Daten in der gleichen Richtung wie der Datenfluss. Im unteren Teil fordert der aktive Empfänger mit einem Methodenaufruf fordere_Daten() beim passiven Sender an. Der Ergebniswert des Methodenaufrufes ist das angeforderte Datenobjekt, das zum Empfänger übertragen wird. Auch in diesem Fall verläuft der Datenfluss vom Sender zum Empfänger. Die Signalisierung zur Datenanforderung erfolgt jedoch in die entgegengesetzte Richtung. Während also der Datenfluss immer vom Sender zum Empfänger verläuft bzw. von der Datenquelle zur Datensenke, kann die Signalisierung auch entgegengesetzt erfolgen.

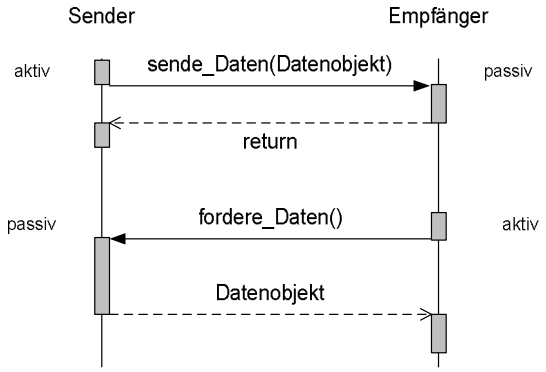


Abbildung 4-38: Sequenzdiagramm der Interaktionen zwischen passiven und aktiven Kommunikationspartnern

Im Modell wird der Datenfluss unabhängig vom Signalisierungsfluss modelliert. Dadurch wird erreicht, dass die anfordernde Komponente von der Komponente verschieden sein kann, welche die angeforderten Daten erhält. Dies erhöht die Kombinierbarkeit von Komponenten im Vergleich zum Architekturmuster „Pipes und Filters“, bei dem Signalisierung und Datenübertragung immer zwischen direkt benachbarten Komponenten erfolgen. Zur Modellierung von Signalisierungsdaten wird der Objekttyp „message“ definiert. Dieser kann beliebige Nachrichten beschreiben, die zwischen Komponenten ausgetauscht werden. Anwendungsentwickler können eigene Subtypen bzw. Kodierungen hinzufügen, um anwendungsspezifische Nachrichten zu beschreiben. Nachrichten werden im Meta-Modell auf die gleiche Weise wie alle weiteren Objekttypen beschrieben. Aufgrund ihrer besonderen Semantik für die Aktivierung von Komponenten und die Steuerung der Verarbeitung im Adaptionsgraph werden Nachrichten jedoch getrennt von den übrigen Objekttypen betrachtet.

Die Trennung von Signalisierung und Datenfluss erfolgt durch die Festlegung, dass Ausgabeports immer aktiv Daten versenden und Eingabeports immer passiv Daten empfangen. Abbildung 4-39 veranschaulicht dies am Beispiel einer Anforderung von Daten zwischen zwei Komponenten. Die Aktivität geht dabei von Komponente 2 aus. Diese erzeugt innerhalb ihrer Anwendungslogik eine Datenanforderung vom Typ (message/request/dataobject). Anforderungen dieses Typs beschreiben ein Datenobjekt anhand eines eindeutigen Identifikators sowie seines Objekttyps. In der Abbildung wird ein Bild vom Typ (image/gif) angefordert. Die Anforderung wird in Form eines Datenobjektes von der Anwendungslogik in Komponente 2 an die Kommunikationslogik übergeben. Diese implementiert die Logik eines Ausgangsports und gibt die Daten entweder an die Logik eines Konnektors oder (wie in der Abbildung) direkt an die Kommunikationslogik eines Eingangsports weiter (1). Die Kommunikationslogik von Komponente 1 empfängt das

Datenobjekt und gibt es an die Verarbeitungslogik weiter. Damit wird Komponente 1 aktiviert. Diese kann nun die Anfrage auswerten und das angeforderte Datenobjekt beispielsweise aus dem Cache laden. Danach wird das ermittelte Datenobjekt an die Kommunikationslogik eines entsprechenden Ausgangsports weitergegeben (2). Dieser vermittelt das Datenobjekt an die Kommunikationslogik eines Eingangsports der Komponente 2. Nach dem Empfang leitet diese das Datenobjekt an die Anwendungslogik weiter, die das Bild verarbeiten und gegebenenfalls weitere Datenobjekte anfordert.

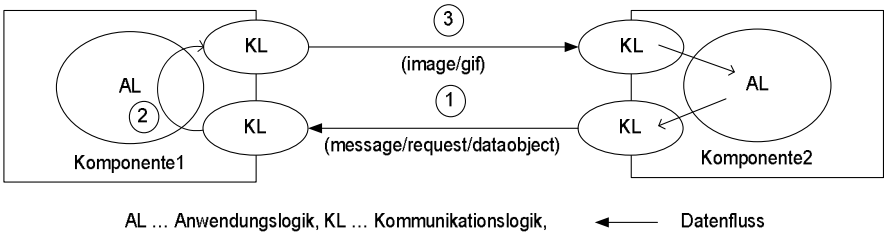


Abbildung 4-39: Anforderung von Daten zwischen Komponenten

Der Sender von Daten ist damit immer der aktive Interaktionspartner. Aus diesem Grund müssen alle Komponententypen außer Datensenken aktiv werden können. Eine Aktivierung kann auf drei Arten erfolgen:

1. *Interne Aktivierung*: Die Komponente besitzt einen eigenen Kontrollfluss (Thread) und wird entsprechend ihrer Implementierung selbst aktiv, d. h. sie fordert Daten an (z. B. periodisch).
2. *Datenaktivierung*: Die Komponente wird durch das Eintreffen eines Datenobjektes an einem Eingangsport aktiviert. Damit wird die Verarbeitung der Komponente angestoßen, deren Resultat an einen Ausgangsport gesendet wird.
3. *Nachrichtenaktivierung*: Die Komponente wird durch das Eintreffen von Nachrichten aktiviert, die ihre Verarbeitung bzw. das Senden von Daten an den Ausgangsport anstoßen. Komponenten benötigen für diese Art der Aktivierung einen Eingangsport der Nachrichten empfangen kann, sowie Code zur Verarbeitung dieser Nachrichten.

Die Aktivierung soll nachfolgend für die einzelnen Komponententypen betrachtet werden.

Datenquellen

Datenquellen erzeugen Anwendungsdaten und stellen somit Anfangspunkte zur Datenübertragung in Adaptionsgraphen dar. Im einfachsten Fall besitzt eine Datenquelle keinen Eingangsport und genau einen Ausgangsport, über den sie aktiv Anwendungsdaten versendet. Dies entspricht dem Verhalten des im linken Teil der Abbildung 4-40 dargestellten Zustandsdiagramms. Eine passive Datenquelle besitzt zusätzlich noch einen Eingangsport zum Empfang von Nachrichten. Das Zustandsdiagramm einer passiven Datenquelle wird im rechten Teil der Abbildung 4-40 dargestellt.

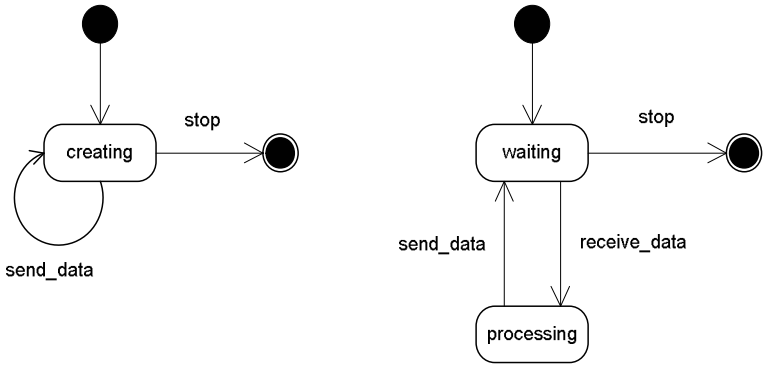


Abbildung 4-40: Zustandsdiagramm einer aktiven (links) und einer passiven Datenquelle (rechts)

Datensenken

Datensenken stellen die Endpunkte von Datenübertragungen in Adaptionsgraphen dar. Dieser Komponententyp entfernt Datenobjekte explizit aus dem Adaptionsgraphen. Datensenken können ebenso wie Datenquellen aktiv oder passiv sein. Der linke Teil der Abbildung 4-41 stellt das Zustandsdiagramm einer aktiven Datensenke dar. Diese erzeugt Anforderungsnachrichten und versendet diese über einen Ausgangsport. Im Gegensatz dazu besitzt eine passive Datensenke nur Eingangsports zum Empfangen von Daten. Ein entsprechendes Zustandsdiagramm beinhaltet der rechte Teil der Abbildung 4-41.

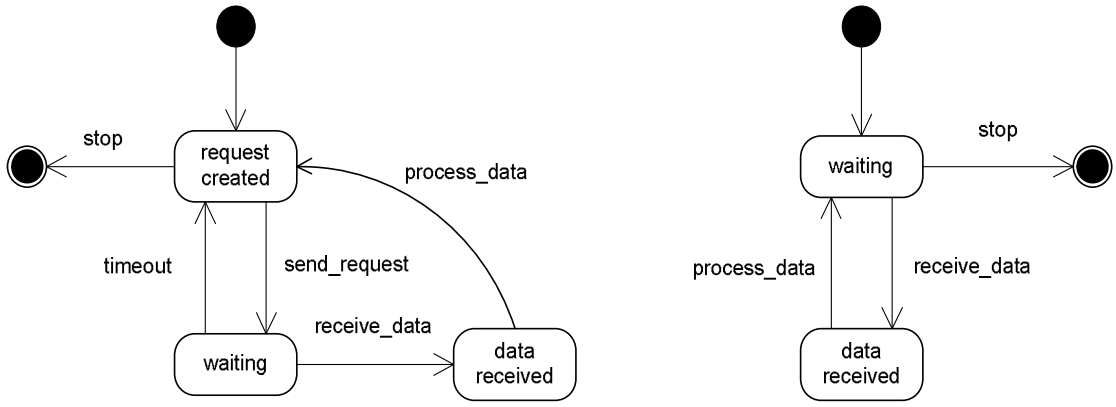


Abbildung 4-41: Zustandsdiagramm einer aktiven (links) und einer passiven Datensenke (rechts)

Verarbeitungskomponenten

In der einfachsten Form wird ein Datenobjekt eines bestimmten Objekttyps unter Berücksichtigung der Komponentenparameter verarbeitet. Die Verarbeitung eines empfangenen Datenobjektes erzeugt genau ein Ergebnisobjekt, das über einen Ausgabeport weitergeleitet wird. Mehrere Ausgangsports können dabei zur Repräsentation verschiedener Ergebnisse der Verarbeitung definiert werden. Beispielsweise können zwei Ausgangsports definiert werden, die eine erfolgreiche Verarbeitung sowie einen Fehler während der Verarbeitung repräsentieren. Nach der Ausgabe eines Datenobjektes über den Fehlerport kann eine weitere Komponente für die Fehlerbehandlung angefügt werden, die unabhängig von der vorhergehenden Komponente arbeiten kann. Verarbeitungskomponenten werden durch das Eintreffen von Datenobjekten aktiviert. Außerdem kann eine Komponente dieses Typs selbst aktiv sein und Daten anfordern. Abbildung 4-42 stellt das Zustandsdiagramm einer passiven Verarbeitungskomponente dar.

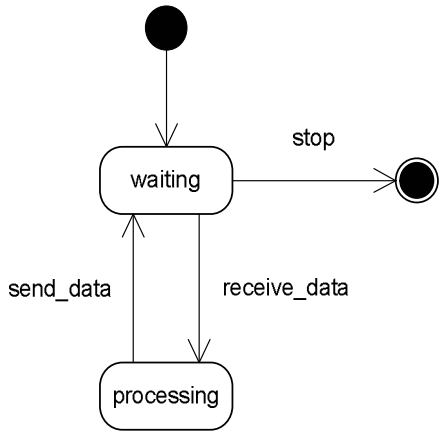


Abbildung 4-42: Zustandsdiagramm einer Verarbeitungskomponente

Sonderfälle von Verarbeitungskomponenten sind die Basismechanismen Komposition und Dekomposition. Beide Mechanismen bilden nicht genau ein Datenobjekt o auf ein Datenobjekt o' ab. Die Dekomposition zerlegt ein Eingangsobjekt o in mehrere Ausgangsobjekte o_1' bis o_n'. Die Komposition fügt mehrere Eingangsobjekte o_1, ... , o_n zu einem Ergebnisobjekt o' zusammen. Die Aktivierung erfolgt jedoch in beiden Fällen wie bei den übrigen Verarbeitungskomponenten durch ankommende Datenobjekte. Bei der Dekomposition resultiert die Verarbeitung jedes Datenobjektes in einer Folge von Ergebnisobjekten. Besitzen die Ergebnisobjekte unterschiedliche Typen, können diese über unterschiedliche Ports ausgegeben werden.

Die Komposition erzeugt dagegen erst dann ein Ergebnisobjekt, wenn alle Teile eines zusammengesetzten Datenobjektes eingetroffen sind. Dazu wird eine Strukturbeschreibung benötigt, anhand derer entschieden werden kann, ob alle Teildaten vorhanden sind. Ist dies der Fall, wird das Ergebnisobjekt erzeugt und weitergesendet. Andernfalls werden die Teildaten in der Komponente zwischengespeichert. Abbildung 4-43 enthält mögliche Zustandsdiagramme für die Komponenten zur Komposition und Dekomposition.

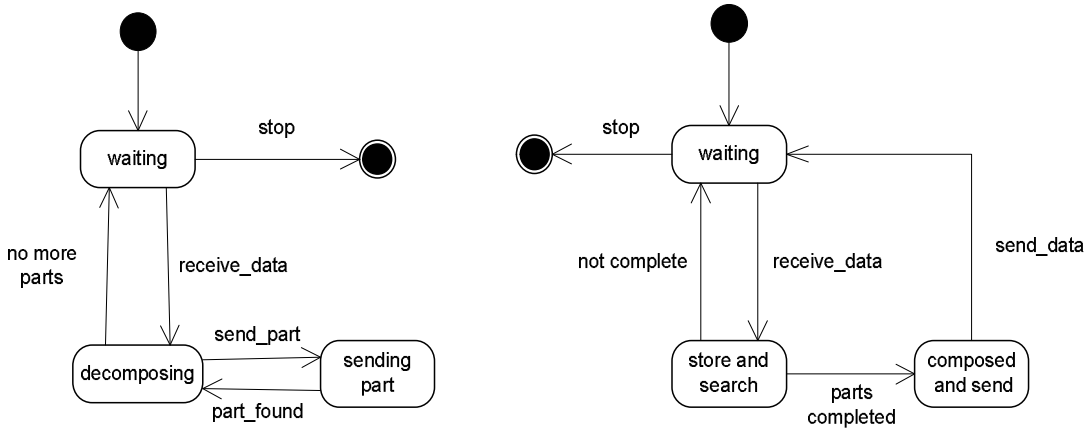


Abbildung 4-43: Zustandsdiagramm einer Dekompositions- (links) und einer Kompositionskomponente (rechts)

Speicherkomponenten

Speicherkomponenten führen im Gegensatz zu Verarbeitungskomponenten zwei Operationen aus: das Speichern und das Suchen von Datenobjekten. Die Anwendungsobjekte wer-

den durch Speicherkomponenten nicht verändert. Beide Operationen arbeiten auf einem gemeinsamen Speicher. Die Operation speichern wird durch das Eintreffen von Datenobjekten aktiviert und speichert diese. Die Operation suchen wird durch eintreffende Nachrichten zur Anforderung von Datenobjekten aktiviert, die ein zu suchendes Datenobjekt beschreiben. Die Beschreibung ist dabei abhängig von der Art des Speichers. Eine Warteschlage benötigt keine Beschreibung des zu suchenden Objektes, da immer das erste Element in der Warteschlange entfernt und weitergeleitet wird. Ein Cache-Speicher benötigt dagegen einen Schlüssel zur Suche eines bestimmten Objektes im Speicher. Abbildung 4-44 stellt ein mögliches Zustandsdiagramm einer Speicherkomponente dar. Alternativ könnte eine Speicherkomponente eine Anforderungsnachricht, die nicht durch Datenobjekte im lokalen Speicher beantwortet werden kann, über einen Nachrichtenport an eine andere Komponente weiterleiten.

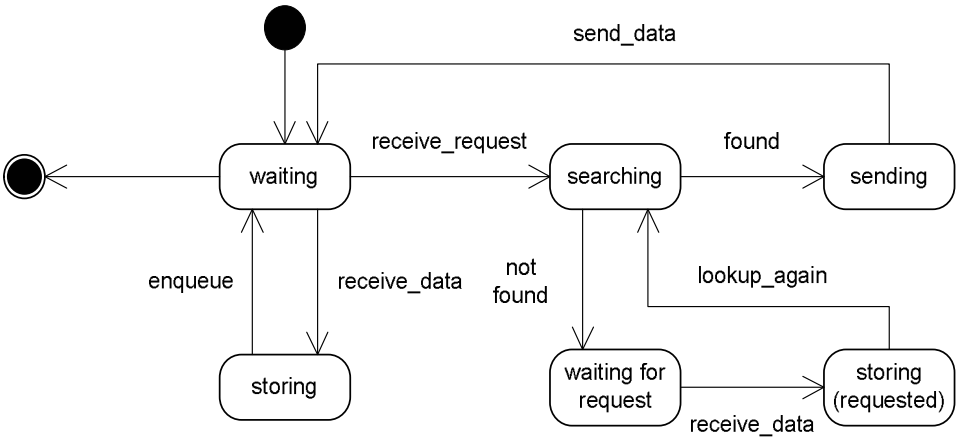


Abbildung 4-44: Zustandsdiagramm einer Speicherkomponente

4.4.6 Zugriff auf Kontext

Der Zugriff auf Kontext wird im Meta-Modell durch abstrakte Sensoren modelliert. Diese kapseln die Details der Kontextermittlung, vor allem die notwendigen Interaktionen mit einem Kontextdienst. Entsprechend Abschnitt 2.4.3 kann Kontext aktiv und passiv genutzt werden. Eine aktive Kontextnutzung entspricht dem Pull-Prinzip, eine passive Kontextnutzung dem Push-Prinzip. Die Art des Zugriffs wird für Sensoren durch den Aktivitätszustand modelliert. Dieser kann die Werte „aktiv“, „passiv“ und „konstant“ annehmen. Die Werte „aktiv“ und „passiv“ korrespondieren mit den Zugriffsarten auf Kontext. Der Wert „konstant“ kennzeichnet den Sensor als Konstante, d. h. der Wert des Sensors wird nicht von einem Kontextdienst ermittelt, sondern bei der Initialisierung des Adaptionspfades zur Laufzeit gesetzt und bleibt dann über die Laufzeit konstant. Das Prinzip der Kontextermittlung, das dem Modell zugrunde liegt, wird in Abbildung 4-45 dargestellt. Diese enthält einen Adaptionsgraph mit zwei Komponenten, die über einen Konnektor verbunden sind. Außerdem besteht der Adaptionsgraph aus vier Sensoren und einer Abbildungsfunktion. Die Sensoren werden in der Abbildung entsprechend ihres Zustandes gekennzeichnet.

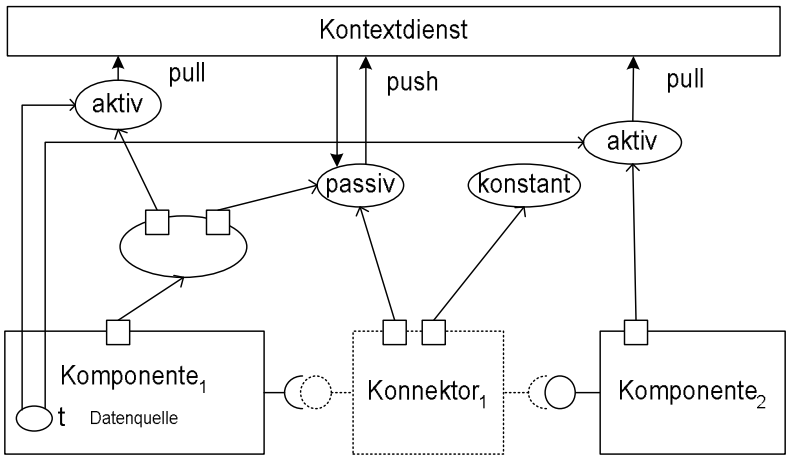


Abbildung 4-45: Prinzip der Kontextermittlung durch Sensoren

Die Kontextermittlung erfolgt abhängig vom Aktivitätszustand eines Sensors nach dem pull- oder push-Prinzip. Die Ermittlung ist jedoch in beiden Fällen zeitlich von den Zugriffen auf Sensoren durch Komponenten bzw. Konnektoren entkoppelt. Passive Sensoren registrieren sich bei ihrer Erzeugung beim Kontextdienst für den repräsentierten Kontextwert und werden bei Änderungen des Wertes aktualisiert. Die Wertermittlung aktiver Sensoren wird mit dem Eintreffen von Datenobjekten assoziiert. Die Ermittlung eines Wertes erfolgt immer dann, wenn ein Datenobjekt in einen Adaptionsgraphen eintritt. Dies muss vom Laufzeitsystem gesteuert werden und ist Teil der Semantik des Komponententyps Datenquelle. Zum Eintrittszeitpunkt t werden demnach die Werte aller aktiven Sensoren vom Kontextdienst ermittelt und mit dem Datenobjekt eindeutig assoziiert. Damit wird die Konsistenz der Kontextwerte aller aktiven Sensoren über die gesamte Verweilzeit eines Datenobjektes in einem Adaptionsgraphen gesichert. Da Datenobjekte nur explizit durch Datensenken aus dem Adaptionsgraphen entfernt werden können, ist gesichert, dass der Zeitpunkt bekannt wird, an dem die Assoziation gelöscht und damit die Kontextwerte verworfen werden können.

Werden Kontextwerte während der Lebenszeit eines Datenobjektes ungültig (z. B. der Verbindungsstatus durch einen Verbindungsabbruch), muss dies vom System erkannt werden. Dies erfolgt durch eine eigenständige Überwachung der Ausführungsumgebung durch das Laufzeitsystem. Welche Kontextwerte überwacht werden müssen, wird durch die aktiven Sensoren der Adaptionsgraphen definiert. Das System kann dann Änderungen beispielsweise in Form von Ereignissen bekannt geben, für die der Entwickler eine Ausnahmebehandlung implementiert.

Der Zugriff auf Sensoren erfolgt innerhalb der Anwendungslogik von Komponenten. Die Anwendungslogik verarbeitet dabei jeweils ein Datenobjekt und greift während der Verarbeitung auf Parameter zu. Bei der Parameterabfrage wird der Identifikator des Datenobjektes übergeben. Dieser erlaubt eine eindeutige Zuordnung von Sensorwerten zu der jeweiligen Abfrage. Parameterzugriffe haben den Zugriff auf Kontextwerte zur Folge, die mit den Parametern durch eine „Zuweisen“-Beziehung verknüpft sind. Die Zugriffe werden über Parameter von Abbildungsfunktionen bis zu den jeweiligen Sensoren weitergeleitet. Dort wird mit Hilfe des Identifikators der Kontextwert ermittelt, der mit dem Datenobjekt assoziiert wurde. Danach erfolgt die Berechnung der Parameter.

Wird der Wert eines Parameters durch einen passiven Sensor zugewiesen, wird bei jedem Zugriff der aktuelle Wert verwendet, unabhängig vom Identifikator des Datenobjektes. In diesem Fall muss der Entwickler die Konsistenz bzw. die Korrektheit der Parameterberechnungen selbst sicherstellen.

4.4.7 Annotationen und Metadaten

Annotationen repräsentieren Meta-Informationen, die an bestimmten Verarbeitungspunkten im Adaptionsgraph konkreten Datenobjekten zugewiesen werden. Die Zuweisung erfolgt durch das Einfügen der Annotationsdaten in den Datencontainer von Datenobjekten. Die Annotationsdaten werden dann gemeinsam mit den Anwendungsdaten im Adaptionsgraphen weitergeleitet. Das Einfügen einer Annotation erfolgt an einem konkreten Verarbeitungspunkt, den ein Datenobjekt durchläuft. Verarbeitungspunkte sind der Verarbeitungslogik von Komponenten zugeordnet. Die Zusammenhänge verdeutlicht Abbildung 4-46. Diese stellt eine Komponente mit je einem Ein- und Ausgangsport sowie einer Adaptionsoperation dar. Der Empfang von Datenobjekten erfolgt über die Verarbeitungslogik des Eingangsports. Dieser übergibt empfangene Datenobjekte an die Anwendungslogik der Komponente. Diese verarbeitet das Datenobjekt und leitet es an die Kommunikationslogik des Ausgangsports weiter. Die Verarbeitungspunkte unterbrechen im Sinne von Interceptoren [OMG01] die Übergabe von Datenobjekten zwischen Kommunikationslogik und Anwendungslogik. Vorverarbeitungspunkte unterbrechen die Übergabe der Datenobjekte vom Eingangsport zur Anwendungslogik, Nachverarbeitungspunkte die Übergabe der verarbeiteten Datenobjekte von der Anwendungslogik zum Ausgangsport. An diesen Punkten wird der Code zum Einfügen bzw. zum Lesen von Annotationen eingefügt. Dieser wird auf allen Datenobjekten ausgeführt, die einen Verarbeitungspunkt durchlaufen. In den Abschnitten 4.4.7.1 und 4.4.7.2 wird die Funktionsweise von Annotationen an zwei Beispielen veranschaulicht.

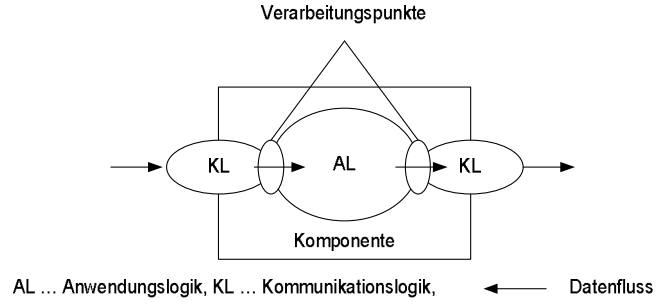


Abbildung 4-46: Verarbeitungspunkte einer Komponente

Die Semantik von Meta-Daten kann ebenfalls anhand der Abbildung 4-46 erläutert werden. An den Verarbeitungspunkten besteht, wie bereits beschrieben, Zugriff auf alle Datenobjekte, die einen Punkt durchlaufen. Damit können neben Annotationen auch weitere Daten aus dem Datencontainer bzw. den enthaltenen Daten selbst gelesen werden. Die Verfügbarkeit dieser Daten wird durch das Meta-Modell in Form von Meta-Daten an den entsprechenden Verarbeitungspunkten explizit modellierbar. Die verfügbaren Meta-Daten können ls Unterklasse von „KontextWert“ Parametern von Abbildungsfunktionen, Komponenten und Konnektoren zugewiesen werden.

4.4.7.1 Beispiel Strommarkierung

In Abbildung 4-47 wird das Markieren von Datenobjekten durch eine Annotation veranschaulicht. Verarbeitungspunkte werden durch Ellipsen dargestellt, die zwischen dem zugehörigen Port und der Komponente eingefügt werden. Eine „Schreiben“-Beziehung wird durch einen gerichteten Pfeil zwischen Verarbeitungspunkt und Annotation in Richtung Annotation dargestellt. Eine „Lesen“-Beziehung wird ebenfalls durch einen gerichte-

ten Pfeil, jedoch in Richtung des Verarbeitungspunktes, notiert. Die Annotation mit dem Identifikator „Strommarkierung“ definiert einen getypten Wert, der die Kennung eines Stromes repräsentiert. Die Abbildung enthält außerdem zwei Kontextwerte zur Konfiguration der Parameter $P_{1.1}$ und $P_{2.1}$. Diese werden durch Ellipsen dargestellt, die durch einen gerichteten Pfeil in Richtung Parameter mit dem konfigurierten Parameter verbunden werden. Die Kontextwerte repräsentieren die Konstanten „1“ und „2“ und definieren über die Parameter die Werte, mit denen die Datenobjekte annotiert werden.

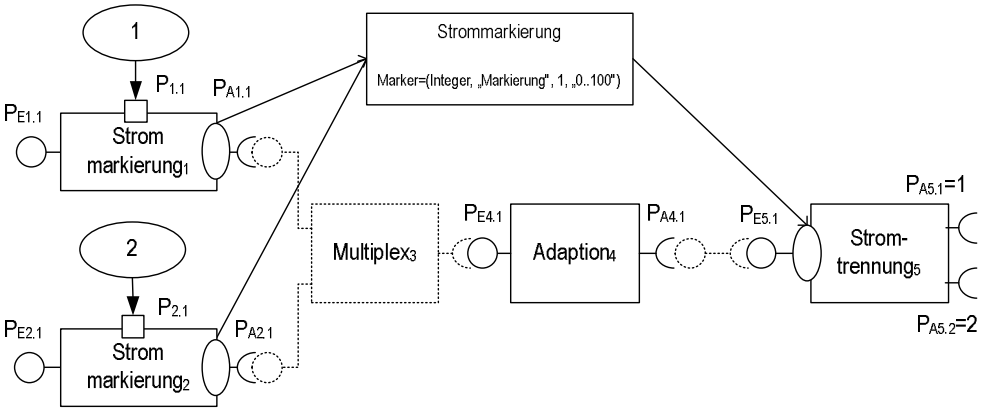


Abbildung 4-47: Prinzip des Softwareroutings unter Verwendung einer Annotation

Die Komponenten $Strommarkierung_1$ und $Strommarkierung_2$ markieren alle empfangenen Datenobjekte mit dem Wert „1“ bzw. „2“. Danach werden diese Datenobjekte zu einem Datenstrom vermischt (Konnektor Multiplex) und können in einem gemeinsamen Komponentenpfad verarbeitet werden. Zur Trennung der beiden Ströme wertet die Komponente „Stromtrennung“ die Annotation aus und versendet die Datenobjekte entsprechend des Wertes der Annotation über zwei verschiedene Ausgangsports. Datenobjekte mit der Kennung „1“ werden über den Ausgangsport $P_{A5.1}$, Datenobjekte mit der Kennung „2“ über den Port $P_{A5.2}$ ausgegeben. Anschließend können die beiden Ströme getrennt weiterverarbeitet werden. Die Markierung von Strömen ermöglicht unter anderem die Mehrfachnutzung von (Teil-) Pfaden, Schleifen in Pfaden, differenzierte Dienste, eine Parallelisierung und Lastverteilung sowie Sitzungen innerhalb eines Adaptionsgraphen.

4.4.7.2 Beispiel Kompositionsinformationen

Die in Abbildung 4-48 dargestellten Komponenten „Dekomposition“ und „Komposition“ führen inverse Funktionen aus. Die Komponente „Dekomposition“ zerlegt ein zusammengesetztes Datenobjekt in eine Folge von Teilobjekten. Die Komponente „Komposition“ stellt das ursprüngliche Datenobjekt wieder her. Nach der Zerlegung des zusammengesetzten Datenobjektes wird der Zusammenhang zwischen den Teilobjekten durch eine Annotation mit dem Identifikatior „Kompositionsinformation“ zu den einzelnen Datenobjekten hinzugefügt. Die Annotation definiert zwei getypte Werte, den Identifikator des zusammengesetzten Datenobjektes (ObjektID) sowie eine Positionskennung der jeweiligen Teildaten innerhalb der ursprünglichen Datenstruktur (SubID). Die Komponente „Dekomposition“ fügt diese Informationen jedem der Teilobjekte hinzu. Die Komponente „Komposition“ kann anhand der annotierten Informationen das Ausgangsobjekt wieder herstellen. Durch die Annotation kann dieser Zusammenhang explizit beschrieben werden.

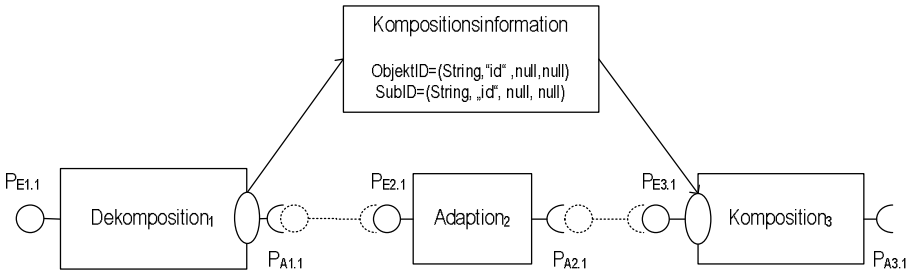


Abbildung 4-48: Eine Stromkennung in Form einer Annotation

Annotationen beschreiben somit Beziehungen zwischen Komponenten in Form von Meta-Informationen. Die Meta-Informationen werden von einer schreibenden Komponente jedem Datenobjekt hinzugefügt, das einen definierten Verarbeitungspunkt durchläuft. Dadurch erfolgt eine eindeutige Zuordnung zwischen Meta-Informationen und Datenobjekten. Die Beziehung zwischen schreibenden und lesenden Komponenten kann explizit und eindeutig beschrieben werden. Durch die Verknüpfung mit Datenobjekten unterliegt die Annotation jedoch auch einigen Einschränkungen. Eine Verteilung der Informationen erfolgt nur in der Richtung des Datenflusses. Außerdem sind die Informationen an den Verarbeitungspfad der Daten gebunden und werden mit den Datenobjekten verzögert. Damit kann eine lesende Komponente überwiegend nicht auf aktuelle Informationen zugreifen (insbesondere, wenn die annotierten Meta-Informationen Zustandsinformationen der schreibenden Komponente enthalten (z. B. die Länge einer Warteschlange). Durch die vielfältige Anwendbarkeit zur Steuerung der Vermittlung von Datenobjekten in Adaptionsgraphen stellt die Annotation jedoch ein wichtiges Konzept zur expliziten Beschreibung dieses Aspektes der Adaption dar.

4.5 Fazit

Als wesentliche Konzepte der Adaption wurden in der vorliegenden Arbeit die Basismechanismen zur Struktur- und Parameteradaption sowie Kontext und Meta-Informationen zur Steuerung der Basismechanismen identifiziert. Diese Konzepte werden durch das spezifizierte Meta-Modell explizit beschreibbar. Basismechanismen zur Parameteradaption sind direkt durch parametrisierbare Komponenten und Konnektoren modellierbar. Strukturänderungen können durch die bedingte Vermittlung von Datenobjekten über alternative Pfade modelliert werden. Damit erfolgt durch das Meta-Modell gemäß Anforderung A4.1 eine Integration von Mechanismen zur Struktur- und Parameteradaption.

Außerdem können entsprechend der Forderung A4.3 Kontext und Meta-Informationen in Form von Sensoren und Annotationen explizit beschrieben werden. Die Abbildung von Kontextwerten auf Parameter wird durch Abbildungsfunktionen ebenfalls explizit modellierbar. Elemente zur bedingten Verzweigung wie der Demultiplex- und der Typsortierkonnektor ermöglichen die Definition von Bedingungen zur Strukturadaption. Auch durch dieses Konzept wird die Anforderung A4.3 erfüllt. Der Forderung nach Plattformunabhängigkeit wird durch die abstrakte Sicht auf adaptive Anwendungen auf der Ebene der Architektur entsprochen (Anforderung A4.4). Das Meta-Modell enthält insbesondere keine plattformabhängigen Elemente. Die Beschreibung der Semantik erfolgte ebenfalls ohne festen Bezug zu plattformspezifischen Konzepten.

Das Meta-Modell enthält außerdem wesentliche Konzepte zur Erhöhung der Flexibilität der Komposition von Komponenten (Anforderung A4.2): Die Vereinheitlichung aller Nachrichten und Anwendungsdaten durch die Konzepte der Datencontainer und Objekt-

typen ermöglicht eine Gleichbehandlung aller Datentypen und eine Verarbeitung dieser in gleichen Komponenten, wenn dies von der Verarbeitungslogik unterstützt wird. Beispielsweise arbeiten Speicherkomponenten unabhängig von Objekttypen und können so Signalisierungsnachrichten und Anwendungsdaten verarbeiten. Die Trennung von Signalisierung und Datenfluss vereinfacht außerdem die Interaktion zwischen Ein- und Ausgangsports auf ein definiertes Muster für alle Komponenten und reduziert damit die Unterschiede zwischen Ports auf die Datenrichtung und die Signatur. Die Aktivität wird von den Ports in die Komponenten verlagert. Ein weiteres wesentliches Konzept ist die Trennung von Ermittlung und Zugriff auf Kontext. Damit wird vollständig von einem Kontextdienst abstrahiert.

Kapitel 5

VALIDIERUNG DES META-MODELLS

In diesem Kapitel wird der Einsatz des Meta-Modells zum Beschreibung adaptiver Anwendungen anhand zahlreicher Beispiele dargestellt. Ziel der Darstellung ist der Nachweis der flexiblen Einsetzbarkeit des Meta-Modells, der Möglichkeiten zur Wiederverwendung einzelner Mechanismen und Teilgraphen sowie der Kombinierbarkeit der Basismechanismen zur Struktur- und Parameteradaption zu adaptiven Anwendungen. Die Beispiele beruhen zum einen auf Ansätzen aus der Literatur, zum anderen auf eigenen Implementierungen (siehe z. B. [SHZ+99, SpS00, SpG02]). Damit sollen sowohl die Dokumentations- und Analysemöglichkeiten der Modellierung als auch die Eignung für den Entwurf adaptiver Anwendungen veranschaulicht werden.

Die Beispielszenarien werden außerdem hinsichtlich der Anforderungen A1.1 bis A1.4 aus Kapitel 1 bewertet. Untersucht wird, welchen der Anforderungen mobiler verteilter Infrastrukturen die einzelnen Adaptionsgraphen genügen. Performancebetrachtungen und die Verbesserung bzw. Optimierung einzelner Mechanismen sind nicht Gegenstand der Beschreibung. Die Validierung wird entsprechend der Klassifikation in die Teile Adaption der Kommunikation, Adaption von Anwendungsdaten und Adaption der Anwendungsstruktur untergliedert. Die Beispiele bauen schrittweise aufeinander auf. Insbesondere können die Beispielgraphen zur Adaption der Kommunikation in die Beispiele zur Datenadaption integriert werden. Mechanismen zur Strukturadaption sind in allen Beispielen enthalten und werden deshalb im dritten Teil nur zusammenfassend betrachtet. Die Notation erfolgt entsprechend der Definitionen aus Kapitel 4. Eine Übersicht der Notation aller Modellierungselemente des Meta-Modells enthält Anhang B.

5.1 Adaption der Datenübertragung

In den folgenden Abschnitten werden (Teil-) Graphen zur Adaption der Datenübertragung beschrieben. Begonnen wird mit einem einfachen Bespiel, das den Einsatz alternativer Kommunikationsprotokolle und die Bedingungen zur Auswahl der einzelnen Protokolle beschreibt. In zwei weiteren Schritten wird dieser Graph zu einem zusammengesetzten Konnektor und zu einer zusammengesetzten Komponente erweitert. Diese Komponente repräsentiert einen Baustein zur Übertragung von Daten zwischen entfernten Rechnern über alternative Kommunikationsprotokolle, der in den weiteren Beispielen wiederverwendet wird. Das zweite Beispiel beschreibt einen Graphen zur Adaption an Verbindungsunterbrechungen. Dieser wird im dritten Beispiel um die Möglichkeit zur prioritätsgesteuerten Übertragung von Datenobjekten erweitert. Aus diesem Teilgraph wird ebenfalls ein wiederverwendbarer Baustein abgeleitet. Der Teilgraph im vierten Beispiel beschreibt eine mögliche Architektur zur Realisierung von abgekoppelten Operationen und integriert die in den vorangegangen Beispielen entwickelten Komponenten und Konnektoren.

5.1.1 Alternative Kommunikationsprotokolle

Der Einsatz alternativer Kommunikationsprotokolle wird in diesem Beispiel auf der Basis des I-TCP Ansatzes (siehe Abschnitt 2.1.2.1) sowie des Ansatzes zur Kompression von TCP-Paketköpfen (siehe Abschnitt 2.1.2.2) modelliert. Abbildung 5-1 stellt einen zusammengesetzten Konnektor dar, mit dem Daten alternativ über eine unveränderte, eine indirekte oder eine TCP-Verbindung mit Kompression der Paketköpfe übertragen werden können. Die indirekte TCP-Verbindung besteht aus zwei unabhängigen Teilverbindungen. Der Zusammenhang zwischen diesen wird entsprechend des I-TCP-Ansatzes durch eine Proxy-Komponente auf der Basisstation hergestellt. Die Komponente empfängt Daten über eine TCP-Verbindung und versendet diese über das Selective Repeat Protocol (SRP) an den Empfänger. SRP ist ein spezielles Transportprotokoll für Kanäle mit hohen Fehlerraten und häufigen gebündelten Fehlern auf Basis von selektiven Bestätigungsnachrichten (siehe [YaB94]). Eine Alternative zu SRP ist das Reliable Data Protocol (RDP), das eine zuverlässige Transportverbindung zwischen mobilem Rechner und der Basisstation herstellt [BaB95b]. Die Kompression von Paketköpfen kann zur Reduzierung der Größe von Bestätigungen zur Unterstützung asymmetrischer Verbindungen eingesetzt werden. Konstante Teile des Paketkopfes der Bestätigungsnachrichten werden dann nur beim Aufbau der Verbindung übertragen und während der Kommunikation aus dem Protokollkopf entfernt. Der entsprechende Konnektor wurde in Abbildung 5-1 mit TCP-HC (mit HC für Header Compression) benannt.

Die Verteilung der Komponenten ist in Abbildung 5-1 dargestellt. Der Sender sowie der Auswahlkonnektor befinden sich auf einem Rechner im Festnetz. Dieser ist über TCP mit der Proxy-Komponente auf der Basisstation verbunden. Die Empfängerkomponente befindet sich auf einem mobilen Rechner und kommuniziert über SRP mit der Basisstation. Alternativ kann eine direkte Verbindung zwischen der Sender- und Empfängerkomponente über TCP-HC bzw. TCP hergestellt werden.

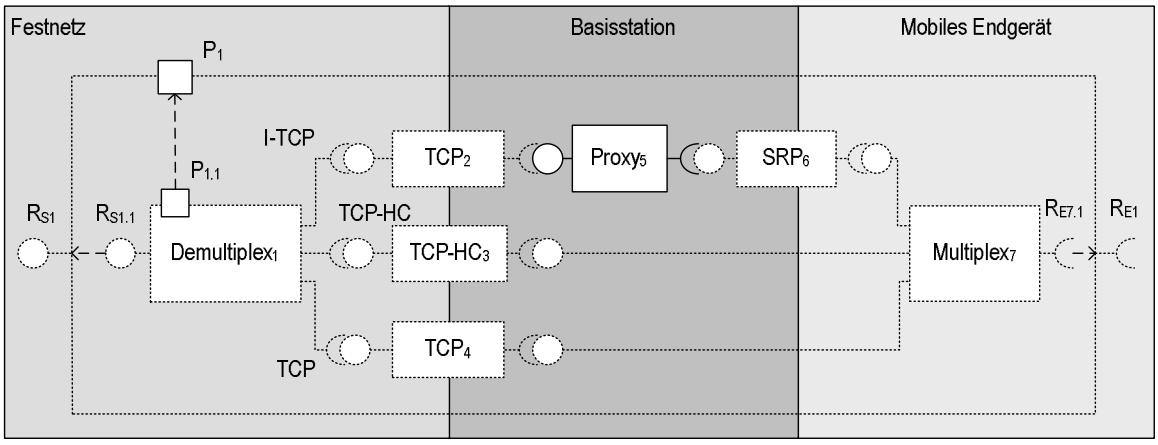


Abbildung 5-1: Die kontextabhängige Auswahl alternativer Kommunikationsprotokolle

Die Entscheidung über die zur Übertragung genutzte Verbindung wird entsprechend der Parametrisierung des Demultiplexkonnektors getroffen. Der Parameter $P_{1.1}$ ist von Typ String und kann entsprechend der Protokollbezeichnungen gesetzt werden. Innerhalb des Konnektors erfolgt dann eine Zuordnung zu den Empfängerrollen, die durch gleichnamige Identifikatoren bezeichnet werden. Dies sind die Identifikatoren „I-TCP“, „TCP-HC“ und „TCP“.

$P_{1.1}$ = {String, “Protokoll”, “TCP”, “{I-TCP, TCP-HC, TCP}“).

Durch den Parameter $P_{1.1}$ kann also die Empfängerrolle direkt benannt und damit das verwendete Protokoll festgelegt werden.

Zur Auswahl des Protokolls können Informationen über die verfügbaren Zugangstechnologien sowie der Aufenthaltsort des Benutzers herangezogen werden. Für das Beispiel werden die folgenden Festlegungen getroffen. Ist ein Kommunikationsmedium mit kleiner Fehlerrate und hoher Datenrate verfügbar (z. B. Ethernet), wird das unveränderte TCP-Protokoll eingesetzt. Ist jedoch nur eine drahtlose Technologie mit hoher Fehlerrate verfügbar, erfolgt die Kommunikation über I-TCP. Dazu müssen aber die Basisstationen eine entsprechende Unterstützung bieten. Im Beispiel wird angenommen, dass aus dem Aufenthaltsort auf die Verfügbarkeit einer Unterstützung für I-TCP geschlossen werden kann. Soll eine Verbindung über einen asymmetrischen Kommunikationskanal aufgebaut werden, wird TCP-HC eingesetzt, um Engpässe durch den schmalbandigen Rückkanal zu vermeiden.

Für die Ermittlung von $P_{1.1}$ wurden dementsprechend die folgenden Sensoren definiert:

S_1=(Verbindung.Zugangsnetzwerk.Typ, String, “Netzwerktyp”, null , {Funk, Infrarot, Festnetz}, passiv)

S_2=(Verbindung.Paketfehlerrate, Float, “”, null, null, passiv)

S_3=(Verbindung.Zugangsnetzwerk.Sendekanal, Float, “bit/s”, null , null, passiv)

S_4=(Verbindung.Zugangsnetzwerk.Rückkanal, Float, “bit/s”, null , null, passiv)

S_5=(Benutzer.Aufenthaltsort.Netzwerk, String, “Netzwerkbezeichner”, null , null, passiv)

Die Abbildungsfunktion AF_1 greift über die fünf Parameter $P_{AF1.1}$ bis $P_{AF1.5}$ auf die Sensoren S_1 bis S_5 zu. Der Resultatwert von AF_1 und damit der Parameter $P_{1.1}$ werden durch die folgenden Regeln definiert:

AF_1=“I-TCP“ if ($P_{AF1.1}$=(“Funk“ or „Infrarot“) and $P_{AF1.2}$>5% and ($P_{AF1.3}$= $P_{AF1.4}$) and $P_{AF1.5}$=“HeimatNetzwerk“)

AF_1=“TCP-HC“ if ($P_{AF1.3}$> $P_{AF1.4}$)

AF_1=“TCP“ else.

Der Sensor S_1 enthält den Typ des Zugangsnetzwerkes. Dieser kann einen der Werte „Festnetz“, „Infrarot“ und „Funk“ annehmen. Der Sensor S_2 enthält die Paketfehlerrate des Netzwerks. Die Sensoren S_3 und S_4 enthalten die Datenrate des Sende- (downlink) bzw. Rückkanals (uplink) und werden genutzt, um symmetrische und asymmetrische Kanäle zu unterscheiden. Der Sensor S_5 enthält das aktuell genutzte Netz. Ist dieses das Heimatnetzwerk des Benutzers, wird für das Beispiel angenommen, dass dort eine Unterstützung für I-TCP auf den Basisstationen zur Verfügung steht.

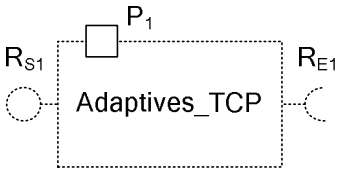


Abbildung 5-2: Notation des zusammengesetzten Konnektors „Adaptives_TCP“

Die Rollen R_{S1} und R_{E1} sowie der Parameter P_1 repräsentieren die Elemente des zusammengesetzten Konnektors „Adaptives_TCP“. Diese werden durch die Abbildung der Elemente $R_{E7.1}$, $R_{S1.1}$ und $P_{1.1}$ im Adaptionsgraphen aus Abbildung 5-1 definiert. Die Notation des zusammengesetzten Konnektors wird in Abbildung 5-2 dargestellt.

5.1.2 Die zusammengesetzte Übertragungskomponente mit Empfangsbestätigung

Der Konnektor „Adaptives-TCP“ wird nachfolgend verwendet, um eine Komponente zur Übertragungssteuerung zu modellieren. Diese besteht aus einer Sender- und Empfängerkomponente die durch einen „Adaptives-TCP“ Konnektor miteinander verbunden sind (siehe Abbildung 5-3). Die Senderkomponente erhält Datenobjekte über den Port $P_{E1.1}$ und gibt diese am Port $P_{A1.1}$ aus. Außerdem wird das Datenobjekt in der Senderkomponente zwischengespeichert. Nach der Ausgabe an Port $P_{A1.1}$ werden die Datenobjekte über den Konnektor „Adaptives-TCP_2“ an den Port $P_{E4.1}$ der Empfängerkomponente vermittelt. Diese gibt die empfangen Datenobjekte am Port $P_{A4.1}$ aus und erzeugt ein Ereignis zur Empfangsbestätigung vom Objekttyp (event/ack/dataobject), das über den Port $P_{A4.2}$ versendet wird. Die Empfangsbestätigung wird über den TCP_3 Konnektor an den Port $P_{E1.2}$ der Senderkomponente vermittelt. Der Sender kann daraufhin das bestätigte Datenobjekt verwerfen und ein neues Datenobjekt anfordern. Dazu sendet er ein Ereignis vom Typ (event/request/next) an den Port $P_{A1.2}$.

Erhält die Senderkomponente innerhalb des gesetzten Timeout-Intervalls keine Bestätigung, testet diese den aktuellen Zustand der Verbindung. Ist die Verbindung verfügbar, wird die Übertragung wiederholt. Bei einem Verbindungsabbruch wird gewartet, bis eine neue Verbindung verfügbar ist, danach wird das Datenobjekt nochmals gesendet. Durch den Parameter $P_{2.1}$ wird das Übertragungsprotokoll für die Übertragung der Datenobjekte entsprechend der Erläuterungen im vorhergehenden Abschnitt festgelegt. Da in der Richtung zum Sender nur Bestätungsnachrichten, also nur Datenobjekte geringer Größe, übertragen werden, wird dafür das unveränderte TCP-Protokoll verwendet. Ebenso könnte jedoch auch eine der Komponenten zur Behandlung von Verbindungsunterbrechungen aus den nächsten Abschnitten verwendet werden, um die Robustheit der Übertragung zu erhöhen.

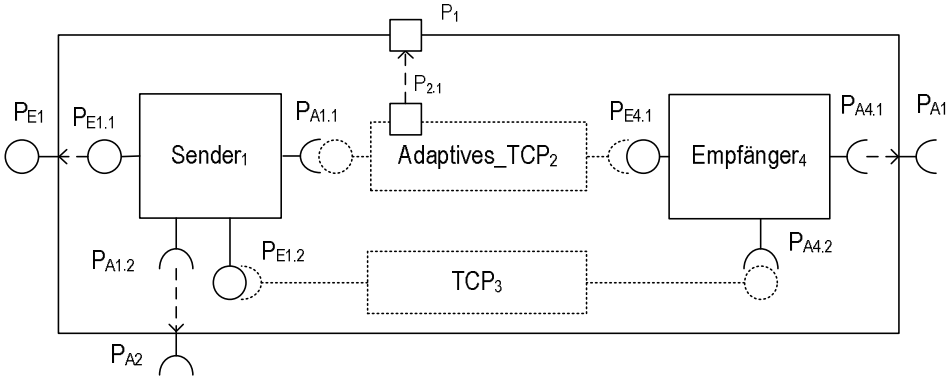


Abbildung 5-3: Definition einer zusammengesetzten Übertragungskomponente mit Empfangsbestätigung

Die Übertragungskomponente in Abbildung 5-4 repräsentiert die zusammengesetzte Komponente entsprechend Abbildung 5-3. Der Parameter P_1 wird durch die Abbildung des Parameters $P_{2.1}$ definiert. Entsprechend erfolgt die Abbildung der Ports $P_{E1.1}$ auf P_{E1}, $P_{A4.1}$ auf P_{A1} und $P_{A1.2}$ auf P_{A2}.

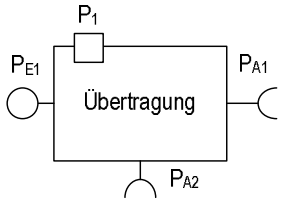


Abbildung 5-4: Notation der Übertragungskomponente

Die Übertragungskomponente wird durch ein eintreffendes Datenobjekt an Port P_{E1} aktiviert. Wurde ein Datenobjekt erfolgreich übertragen, fordert die Übertragungskomponente über den Port P_{A2} ein weiteres Datenobjekt zur Übertragung an und wartet anschließend auf das Eintreffen des nächsten Datenobjektes. Bei einer kurzzeitigen Verbindungsunterbrechung bleibt das Datenobjekt in der Senderkomponente gespeichert und kann erneut übertragen werden, wenn eine neue Verbindung hergestellt wurde.

5.1.3 Adaption an häufige Verbindungsunterbrechungen

Die Komponente „Verzögerte Übertragung“ unterstützt eine Adaption an Verbindungsunterbrechungen und Abkopplungen. Die Modellierung greift dazu das Konzept des Q-RPC des Rover-Projektes auf (siehe Abschnitt 2.1.2.3). RPC-Requests werden in Rover zunächst in einem Protokoll (operation log) gespeichert und durch einen Network Scheduler übertragen, wenn eine entsprechende Verbindung verfügbar ist. Das Protokoll wird durch die Komponente „Warteschlange“ modelliert (siehe Abbildung 5-5). In diese können Datenobjekte über den Port $P_{E2.1}$ eingestellt und unter Beibehaltung der Ankunftsreihenfolge wieder entnommen werden. Die Entnahme erfolgt durch das Senden eines Ereignisses vom Typ (event/request/next) an den Port $P_{E2.2}$. Ist mindestens ein Datenobjekt in der Warteschlange enthalten, wird das entsprechend der Ankunftsreihenfolge nächste Objekt über den Ausgangsport $P_{A2.1}$ versendet. Ist die Warteschlange leer, wird das zuvor empfangene Ereignis über den Ausgangsport $P_{A2.2}$ ausgegeben.

Die Funktion des Network Schedulers wird durch die Komponenten „Steuerung“ und „Transfer“ beschrieben (siehe Abbildung 5-5). Die Komponente „Steuerung“ empfängt Datenobjekte über den Port $P_{E1.1}$ und vermittelt diese entsprechend ihres Zustandes an einen der beiden Ausgabeports $P_{A1.1}$ bzw. $P_{A1.2}$ weiter. Die Komponente kann sich in einem der beiden Zustände „Warteschlange leer“ und „Warteschlange füllen“ befinden. Initial arbeitet die Komponente im Zustand „Warteschlange leer“. Das erste eintreffende Datenobjekt wird deshalb über $P_{A1.2}$ direkt an die Übertragungskomponente gesendet, um diese zu aktivieren. Nach dem Versenden eines Datenobjektes über $P_{A1.2}$ geht die Komponente in den Zustand „Warteschlange füllen“ über. In diesem werden alle eintreffenden Datenobjekte über den Ausgangsport $P_{A1.1}$ versendet. Eine Rückkehr in den Zustand „Warteschlange leer“ erfolgt erst, wenn alle Datenobjekte aus der Warteschlange entfernt wurden. Dies wird durch das Eintreffen eines Ereignisses vom Typ (event/request/next) am Eingangsport $P_{E1.2}$ signalisiert, das von der Warteschlangenkomponente über den Ausgangsport $P_{A2.2}$ weitergeleitet wird, wenn die Warteschlange leer ist.

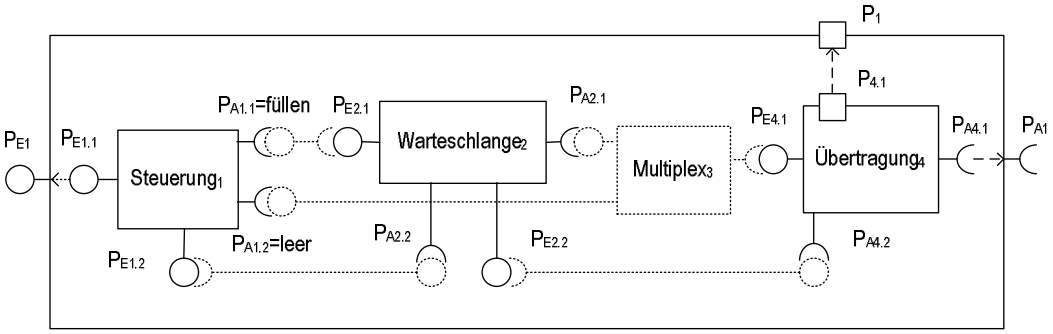


Abbildung 5-5: Adaptionsgraph mit einer Warteschlangenkomponenten zur Unterstützung von Verbindungsunterbrechungen

Die Komponente „Übertragung“ arbeitet wie im vorherigen Abschnitt beschrieben und vermittelt die Datenobjekte über das konfigurierte Protokoll. Abbildung 5-6 enthält die Notation der zusammengesetzten Komponente „Verzögerte Übertragung“.

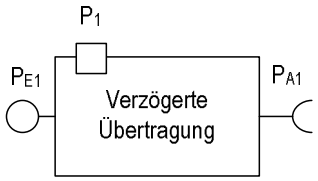


Abbildung 5-6: Notation der Komponente „Verzögerte Übertragung“

Die Behandlung kurzzeitiger Verbindungsabbrüche erfolgt durch die Übertragungskomponente, die bei einer Verbindungsunterbrechung die Übertragung wiederholt. Nach dem erfolgreichen Versenden eines Datenobjektes wird dann das nächste Datenobjekt angefordert. Bei längeren Zeiten der Abkopplung können durch die Komponente zur verzögerten Übertragung dennoch Datenobjekte in Empfang genommen werden, da diese in der Warteschlangenkomponente zwischengespeichert werden. Nach dem Aufbau einer neuen Verbindung wird die Warteschlange durch die Übertragungskomponente geleert.

5.1.4 Priorisierung von Datenströmen

In dem in Abschnitt 5.1.3 beschriebenen Graphen werden alle Datenobjekte entsprechend ihrer Ankunftsreihenfolge übertragen. Durch eine Priorisierung von Datenobjekten kann eine Optimierung der Nutzung einer Verbindung erreicht werden, insbesondere wenn diese nicht nur durch häufige Verbindungsunterbrechungen und Phasen der Abkopplung, sondern auch durch eine limitierte Datenrate gekennzeichnet ist. Eine Lösungsmöglichkeit dafür ist die Priorisierung von Daten, die beispielsweise im Trickle-Reintegration Verfahren in CODA und in Verbindung mit Queued-RPCs in Rover eingesetzt wird. Beide Ansätze unterscheiden entsprechend der Bedeutung der Daten für ein Weiterarbeiten auf dem mobilen Rechner mehrere Klassen von Daten (siehe Abschnitt 2.1.1.2 und 2.1.2.3).

Abbildung 5-7 enthält einen Graphen zur Modellierung einer Prioritätswarteschlange durch die Erweiterung des Graphen aus Abbildung 5-5. Die unterschiedliche Behandlung der Datenobjekte entsprechend ihrer Priorität wird durch die Kopplung mehrerer Warteschlangenkomponenten zu einer Prioritätswarteschlange erreicht. Über den Eingangsport $P_{E7.2}$ kann wie bei der Komponente zur verzögerten Übertragung durch ein Ereignis des Typs (event/request/next) das nächste Datenobjekt aus der Warteschlange entnommen werden. Ist die höher priorisierte Warteschlange ($Warteschlange_7$) leer, vermittelt sie das Ereignis

über den Ausgangsport $P_{A7.2}$ an den Eingangsport $P_{E6.2}$ der zweiten Warteschlange (Warteschlange$_6$) weiter. Ist diese Warteschlange ebenfalls leer, erhält die Steuerungskomponente (Steuerung$_4$) über den Ausgangsport $P_{A6.2}$ und den Eingangsport $P_{E4.2}$ das entsprechende Ereignis. Der Zustand „Warteschlange leer“ beschreibt dabei den Zustand aller Warteschlangen. In diesem Zustand sendet die Steuerungskomponente das erste eintreffende Datenobjekt über $P_{A4.2}$ über den Multiplexkonnektor (Multiplex$_8$) zu einem mit der Empfängerrolle $R_{E8.1}$ zu verknüpfenden Ausgangsport. Nach dem Zustandsübergang zu „Warteschlange füllen“ werden alle Datenobjekte an den Ausgangsport $P_{A4.1}$ vermittelt. Die Steuerungskomponente kann dabei unverändert aus Abbildung 5-5 übernommen werden. Die Anzahl der Warteschlangen, für die der Zugriff gesteuert wird, bleibt vor der Steuerungskomponente verborgen.

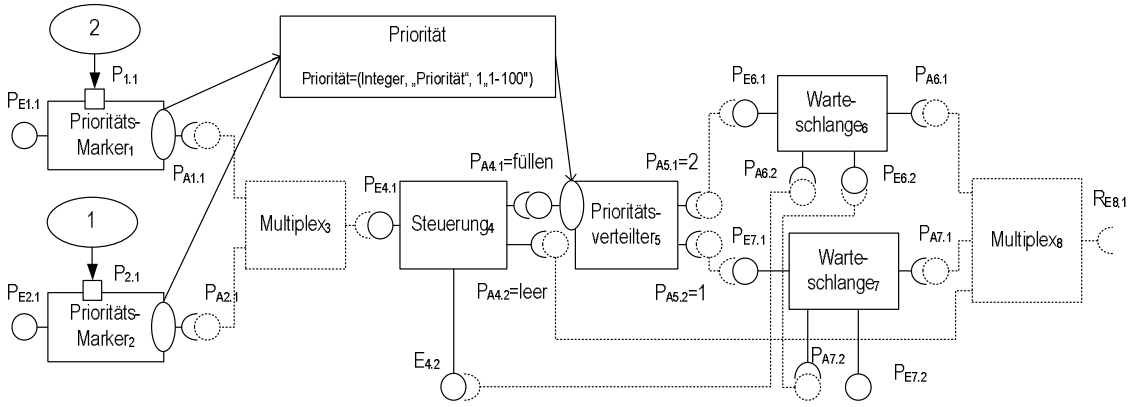


Abbildung 5-7: Adaptionsgraph zur Priorisierung von Datenströmen

Die Vergabe der Prioritäten erfolgt über die Komponente „PrioritätsMarker“. Diese weisen, gemäß dem Mechanismus zur Annotation, jedem eintreffenden Datenobjekt eine Priorität zu. Die zugewiesenen Prioritäten werden über die Parameter $P_{1.1}$ und $P_{2.1}$ konfiguriert. Demnach erhalten über den Eingangsport $P_{E1.1}$ empfangene Datenobjekte die Priorität 2 und über den Eingangsport $P_{E2.1}$ empfangen Datenobjekte die Priorität 1. Entsprechend der Konfiguration repräsentiert ein kleinerer Wert eine höhere Priorität. Mit der eingefügten Annotation werden die Datenobjekte über den ersten Multiplexkonnektor (Multiplex$_3$) und die Steuerungskomponente an den Prioritätsverteiler (Prioritätsverteiler$_5$) vermittelt. Diese Komponente greift lesend auf die Annotation „Priorität“ zu und vermittelt die Datenobjekte entsprechend dieser Information an einen ihrer Ausgangsports $P_{A5.1}$ (Priorität 2) und $P_{A5.2}$ (Priorität 1).

Über den Eingangsport $P_{E7.2}$ ankommende Anforderungen werden zuerst aus der ersten Warteschlange (Warteschlange$_7$) bedient. Erst wenn diese leer ist, wird die Anforderung an die zweite Warteschlange (Warteschlange$_6$) vermittelt. Entnommene Datenobjekte werden über den zweiten Multiplexkonnektor (Multiplex$_8$) gesendet.

Die Komponente in Abbildung 5-8 definiert eine Prioritätswarteschlange. Die Anzahl der unterschiedenen Prioritäten kann durch Hinzufügen von Warteschlangen- und Prioritäts-Marker-Komponenten nach dem Muster aus Abbildung 5-7 erfolgen. Die Ports $P_{E1.1}$, $P_{E2.1}$, $P_{E7.2}$ werden entsprechend auf die Ports P_{E1}, P_{E2}, P_{E3} der zusammengesetzten Komponente abgebildet. Die Empfängerrolle $R_{E8.1}$ muss zuvor mit einer Komponente verbunden werden, deren Ausgangsport auf P_{A1} abgebildet werden kann. Dazu kann eine NOP-Komponente als Hilfskomponente herangezogen werden, die keine Verarbeitungsoperation auf eintreffenden Datenobjekten ausführt, sondern diese nur an einen Ausgangsport vermittelt. Außerdem wird die Annotation „Priorität“ auf den Verarbeitungspunkt (Nachver-

arbeitung) des Ports P_{A1} abgebildet. Entsprechend der im Meta-Modell definierten Semantik enthalten damit alle Datenobjekte, die über den Port P_{A1} ausgegeben werden, eine annotierte Priorität.

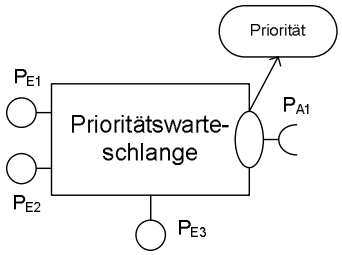


Abbildung 5-8: Notation der Prioritätswarteschlange

Die Prioritätswarteschlange ist ein Beispiel dafür, wie im Modell die Funktionalität einer Komponente in Basiskomponenten zerlegt werden kann. Eine Prioritätswarteschlange wird in einer Implementierung in der Regel durch eine atomare Komponente realisiert werden. Die weitere Zerlegung im Modell kann jedoch Details einer solchen Implementierung beschreiben. Außerdem kann durch das Hinzufügen und Ersetzen von Komponenten eine Veränderung der Funktionalität, beispielsweise der Steuerung der Warteschlangen, beschrieben werden. Die Zerlegung trägt damit vor allem zum Verständnis der Funktionalität von Komponenten bei und unterstützt Entwickler während des Entwurfs adaptiver Komponenten, für den eine Zerlegung zur Identifikation von Parametern und Ports sehr hilfreich sein kann.

5.1.5 Adaptionsgraphen für abgekoppelte Operationen

Aufbauend auf den in den vorangegangenen Abschnitten erarbeiteten Komponenten sollen im folgenden Beispiel abgekoppelte Operationen als Adaptionsgraph modelliert werden. Die Betrachtung geht davon aus, dass die Anwendung nur lesend auf die Daten zugreift. Der Graph beschreibt deshalb nur das Vorabladen und Zwischenspeichern von Daten auf dem Endgerät, um auch in Phasen der Abkopplung auf diese Daten zugreifen zu können. Außerdem erfolgt eine Priorisierung der Anforderungen. Direkt von der Anwendung gesendete Anforderungen haben eine höhere Priorität als die Anforderungen zum Vorabladen von Daten. Einen entsprechenden Graphen zur Realisierung des Bespielszenarios enthält Abbildung 5-9.

In diesem Szenario befinden sich die Komponenten zum Vorabladen (Vorabladesteuerung$_8$) und zum zwischenspeichern von Daten (Cache$_6$) auf dem mobilen Endgerät, um ein abgekoppeltes Arbeiten zu ermöglichen. Über den Eingangsport $P_{E6.2}$ kann eine Client-Komponente Datenobjekte anfordern. Die Anforderung wird durch die Cachekomponente verarbeitet und aus dem Cache beantwortet, wenn das Datenobjekt dort vorliegt. Ist dies nicht der Fall, wird die Anforderung über den Ausgangsport $P_{A6.2}$ an den Eingangsport $P_{E9.2}$ der Prioritätswarteschlange (Prioritätswarteschlange$_9$) weitergegeben. Entsprechend der Definition in Abschnitt 5.1.4 erhalten alle eintreffenden Datenobjekte an diesem Port den Prioritätwert „1“ und damit eine höhere Priorität als Datenobjekte, die an Port $P_{E9.1}$ eintreffen. Die Übertragung wird durch die Übertragungskomponente (Übertragung$_{10}$) entsprechend der Erläuterungen in Abschnitt 5.1.3 gesteuert. Die Komponente „Prioritätsverteiler“ wertet die Priorität der Datenobjekte aus (Annotation Priorität) und trennt die Datenobjekte entsprechend. Datenobjekte mit der Priorität „1“ werden über Port $P_{A11.2}$, Datenobjekte mit der Priorität „2“ über den Port $P_{A11.1}$ ausgegeben. Die von einem Client

explizit gesendeten Anforderungen werden demnach über Port $P_{A11.2}$ ausgegeben und können beispielsweise an eine Serverkomponente weitervermittelt werden.

Die zweite Quelle für Datenanforderungen ist die Komponente „Vorabladesteuerung“. Diese wird in den Pfad der Datenobjekte eingefügt, die explizit angefordert wurden. Aus diesen kann sie entsprechend konfigurierter Regeln oder einer explizit definierten Liste (über Parameter $P_{8.1}$) zukünftige Datenanforderungen ableiten. Diese Daten werden über Port $P_{A8.2}$ vorab angefordert. Die Übertragung der Anforderungen erfolgt ebenfalls über die Prioritätswarteschlange sowie die Übertragungs- und Prioritätsverteilerkomponente. Den Anforderungen zum Vorabladen von Daten wird an Port $P_{E9.1}$ die Priorität „2“ zugewiesen. Diese besitzen damit eine niedrigere Priorität als die expliziten Anforderungen. In der Komponente „Prioritätsverteiler“ werden die Anforderungen zum Vorabladen über Port $P_{A11.1}$ an die VorabladeMarkerkomponente vermittelt. Diese speichert alle eintreffenden Anforderungen und leitet diese Anschließend über Port $P_{A1.2}$ weiter. Dieser Port kann beispielsweise mit einem entsprechenden Port zur Datenanforderung einer Serverkomponente verbunden werden.

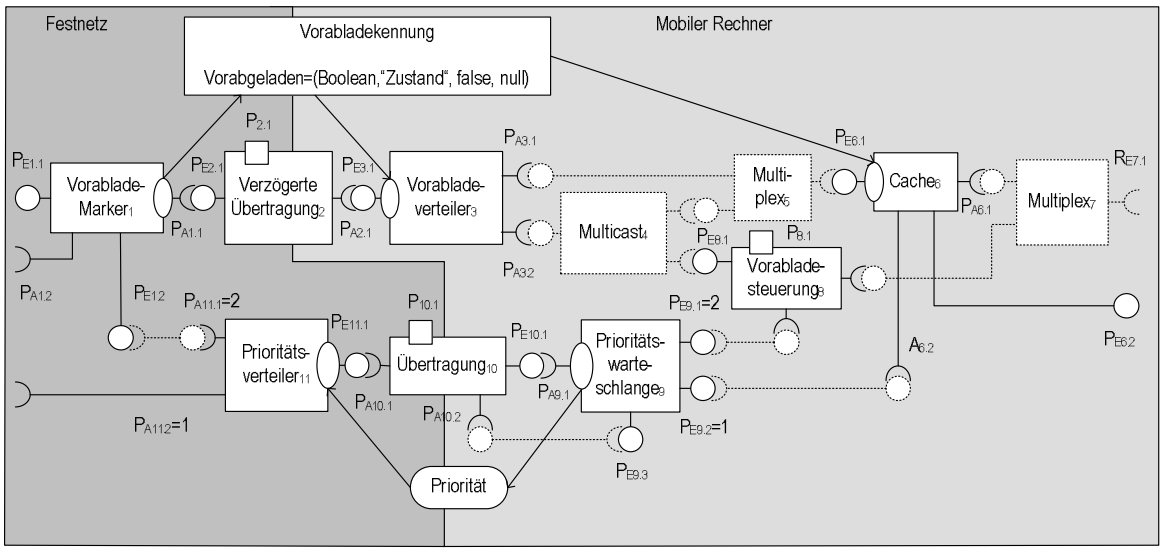


Abbildung 5-9: Adaptionsgraph für abgekoppelte Operationen

Die angeforderten Daten treffen dann über Port $P_{E1.1}$ bei der VorabladeMarkerkomponente ein. Diese markiert entsprechend der vorliegenden Vorablade-Anforderungen die vorab geladenen Datenobjekte (Annotation Vorabladekennung) und versendet alle erhaltenen Datenobjekte über Port $P_{A1.1}$. Die Komponente zur verzögerten Übertragung (siehe Abschnitt 5.1.3) überträgt die Daten über das Netzwerk und gibt diese über Port $P_{A2.1}$ und $P_{E3.1}$ an die Komponente Vorabladeverteiler weiter. Diese trennt die Daten entsprechend der Vorabladekennung in vorab geladene (Port $P_{A3.1}$) und explizit angeforderte Datenobjekte (Port $P_{A3.2}$). Vorab geladene Datenobjekte werden über den Multiplexkonnektor ($Multiplex_5$) an den Cache weitergeleitet und dort gespeichert. Der Cache verwendet die Vorabladekennung zur Steuerung der Verdrängung von zwischengespeicherten Datenobjekten. Explizit angeforderte Datenobjekte werden dagegen per Multicast sowohl im Cache gespeichert, als auch über die Komponente „Vorabladesteuerung“ an die Empfängerrolle $R_{E7.1}$ des Multiplexkonnektors ($Multiplex_7$) weitergeleitet. Die Empfängerrolle kann dann mit dem Eingangsport des Clients verbunden werden, der die Daten angefordert hat.

Der Graph zum Vorabladen von Daten kann in verschiedene Anwendungen integriert werden. Die Komponente zur Vorabladesteuerung muss dabei anwendungsspezifische

Datenanforderungen erzeugen können. Außerdem muss die Komponente „Vorabladeverteiler“ diesen Anforderungen eintreffende Datenobjekte zuordnen können. Alle weiteren Komponenten sind unabhängig von der Anwendung. Insbesondere erfolgt das Vorabladen transparent für Client und Server einer Anwendung. Der Graph unterstützt weiterhin die priorisierte Übertragung expliziter Anforderungen und die Behandlung von kurzzeitigen Verbindungsunterbrechungen sowie längere Phasen der Abkopplung. Die Parameter $P_{2.1}$ und $P_{10.1}$ erlauben dabei die adaptive Auswahl eines der Protokolle, die von den Komponenten „Übertragung“ und „Verzögerte Übertragung“ unterstützt werden. Der Struktur des Teilgraphen liegen die Erkenntnisse aus Abschnitt 2.1.1 zugrunde. Der Parameter $P_{8.1}$ der Komponente zur Vorabladesteuerung kann entweder explizit durch den Anwender in Form einer Liste (Hoarding) oder mit Regeln zur Ableitung vorabzuladender Daten aus den aktuell verarbeiteten Daten definiert werden (Prefetching).

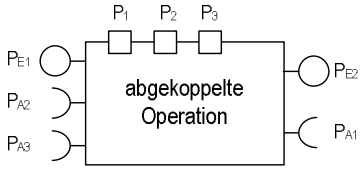


Abbildung 5-10: Zusammengesetzte Komponente zur Realisierung abgekoppelter Operationen

Abbildung 5-10 stellt die zusammengesetzte Komponente zur Ausführung abgekoppelter Operationen dar. Diese wird durch die Abbildung der Ports von Komponenten in Abbildung 5-9 definiert. Dabei wird Port $P_{E1.1}$ auf P_{E1}, $P_{A1.2}$ auf P_{A2}, $P_{A11.2}$ auf P_{A3} und $P_{E6.2}$ auf P_{E2} abgebildet. Außerdem wird die Empfängerrolle $R_{E7.1}$ mit einer NOP-Komponente verbunden, deren Ausgabeport auf den Port P_{A1} abgebildet wird. Die Parameter $P_{2.1}$ und $P_{10.1}$ werden auf die Parameter P_1 bzw. P_2 abgebildet und können zur Definition des Übertragungsprotokolls in den Komponenten „Übertragung“ und „Verzögerte Übertragung“ verwendet werden. Die Komponente „abgekoppelteOperation“ empfängt über den Eingabeport P_{E2} Anforderungen zum Laden von Daten. Die Anforderungen werden entsprechend Abbildung 5-9 entweder aus dem Cache direkt beantwortet oder an den Ausgabeport P_{A3} vermittelt. Datenobjekte können über den Eingangsport P_{E1} zur Vermittlung durch die Komponente übergeben werden. Die Daten werden durch die Komponente übertragen, im Cache zwischengespeichert und am Ausgabeport P_{A1} ausgegeben. Außerdem werden die Dateobjekte von der Komponente-Vorabladesteuerung in Abbildung 5-9 untersucht. Diese erzeugt dann entsprechend ihrer Konfiguration über Parameter $P_{8.1}$ (P_3) Anforderungen zum Vorabladen von Datenobjekten. Diese werden mit geringerer Priorität als die direkten Anforderungen übertragen und über den Ausgabeport P_{A2} ausgegeben.

5.2 Adaption von Anwendungunsdaten

Die zusammengesetzten Elemente und Adaptionsgraphen der folgenden Abschnitte modellieren die Adaption von Anwendungsdaten. Auch diese Mechanismen werden schrittweise zu komplexeren Beispielen zusammengesetzt. Es erfolgt außerdem eine Kombination mit Mechanismen zur Adaption der Datenübertragung. Begonnen wird mit einem Bespiel zur Filterung von E-Mail-Nachrichten. Dieses beschreibt außerdem die Grundstruktur einer E-Mail-Anwendung, in die in den weiteren Beispielen zusätzliche Mechanismen eingefügt werden.

5.2.1 Filterung von Daten

Als erstes Beispiel soll das Auswählen und Verwerfen von Anwendungsdaten betrachtet werden. Dies erfolgt anhand einer E-Mail-Anwendung, deren Grundstruktur in Abbildung 5-11 dargestellt wird. Demnach besteht die Anwendung aus einer Datenquelle und einer Datensenke. Die Datenquelle modelliert einen Mail-Server, der um eine aktive Abfragekomponente erweitert wurde (zusammengefasst in der Komponente „MailSource“). Diese fragt Nachrichten periodisch ab und sendet neue Nachrichten proaktiv an den E-Mail-Client. Außerdem kann der E-Mail-Client auch selbst Nachrichten anfordern. Dazu kann er konkrete Nachrichten auswählen bzw. Regeln definieren (z. B. alle Nachrichten von Sender X; alle Nachrichten, die nicht älter als Y Tage sind).

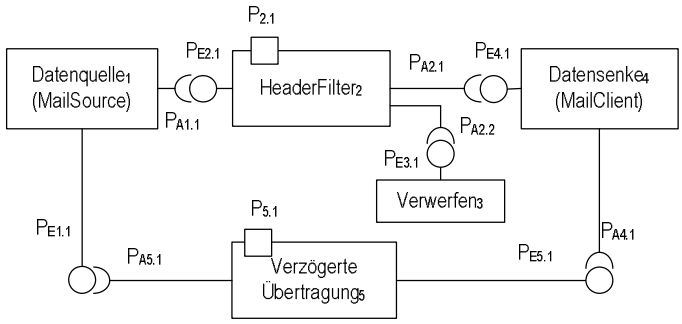


Abbildung 5-11: Grundstruktur der E-Mail-Anwendung

Im folgenden Beispiel soll zunächst die Filterung neuer Nachrichten beschrieben werden, die von der Komponente „MailSource“ proaktiv an die Komponente „MailClient“ gesendet werden. Diese werden am Ausgangsport $P_{A1.1}$ ausgegeben und an den Eingangsport $P_{E2.1}$ vermittelt. Die Komponente „HeaderFilter“ analysiert die E-Mail-Nachrichten anhand einer Menge von Filterausdrücken, die über den Parameter $P_{2.1}$ definiert werden. Dieser ist vom Typ Set(String). Die einzelnen Zeichenketten definieren Ausdrücke der Form (header; Zeichenkette). Beispielsweise definieren die folgenden Muster, das nur Nachrichten der angegebenen E-Mail-Adresse oder mit dem Thema „Projekt“ weitergeleitet werden sollen:

$P_{2.1}$=Set{„from; springet@rn.inf.tu-dresden.de“, „subject; Projekt“}

Die Definition und die Interpretation der Ausdrücke sind implementierungsspezifisch. Beispielsweise könnten auch ausschließende Filter oder logische Ausdrücke als Filterregeln definiert werden. Ebenso können entsprechende Regeln zur Realisierung eines Spam-Filters definiert werden. Durch die Verwendung einer Abbildungsfunktion können auch alternative Regelmengen definiert werden, die in Abhängigkeit des jeweiligen Umgebungszustandes angewendet werden (z. B. in Abhängigkeit von der verfügbaren Datenrate oder der Endgeräteklasse). So könnten bei einer geringen Datenrate wesentlich restriktivere Regeln angewendet werden, als bei einer hohen Datenrate. Im Beispiel entsprechend Abbildung 5-11 werden alle Nachrichten die mindestens eine der Regeln in der Liste erfüllen über den Ausgangsport $P_{A2.1}$ an die MailClient-Komponente weitergesendet. Alle weiteren Nachrichten werden über Port $P_{A2.2}$ ausgegeben und in der Komponente „Verwerfen“ aus dem Adaptionsgraphen entfernt.

5.2.2 Zerlegen und Zusammensetzen von strukturierten Datenobjekten

Im ersten Beispiel wurde die Grundstruktur zur Modellierung einer adaptiven E-Mail-Anwendung vorgestellt, die in Abhängigkeit von Kontext unterschiedlich restriktive Filterregeln anwendet, um die Anzahl der Nachrichten und damit auch die über das Netzwerk zu

übertragende Datenmenge zu steuern. Die Auswahl der zu verwerfenden Nachrichten erfolgt durch standard- oder benutzerdefinierte Regeln. Damit ist der Informationsverlust durch den Benutzer auf Basis ganzer Nachrichten und deren Headerinformationen möglich. Im zweiten Beispiel wird ein Basisgraph zur fein-granularen Adaption von Nachrichtenteilen auf der Basis von Transformationsmechanismen beschrieben. Der Graph besteht aus zwei Komponenten, die eine Transformation strukturierter E-Mail-Nachrichten ausführen (siehe Abbildung 5-12).

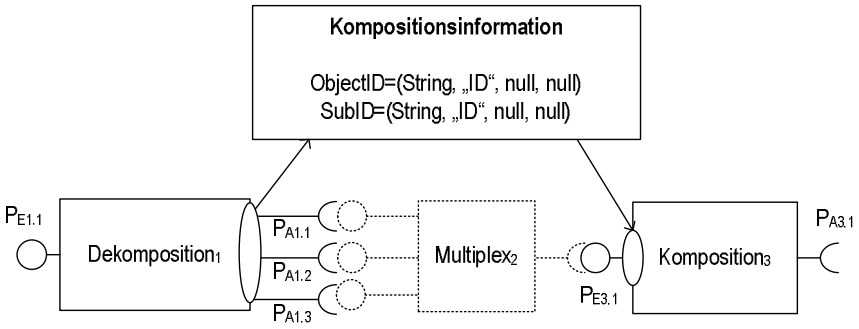


Abbildung 5-12: Adaptionsgraph für E-Mail-Nachrichten

Die Dekompositionskomponente zerlegt eine Nachricht in die Bestandteile Nachrichtenkopf, Nachrichtentext und Anhänge. Der Nachrichtenkopf enthält zusätzlich die Informationen über die Anzahl der Bestandteile der Nachricht. Diese Bestandteile werden in einer Sequenz an den drei Ausgangsports $P_{A1.1}$ (Nachrichtenkopf), $P_{A1.2}$ (Nachrichtentext) und $P_{A1.3}$ (Anhänge) ausgegeben. Allen Bestandteilen wird außerdem eine Annotation „Kompositionsinformation“ hinzugefügt, wie dies bereits in Abschnitt 4.4.7.1 erläutert wurde. Im vorliegenden Beispiel erfolgt keine Adaption der einzelnen Datenobjekte. Diese werden durch den Multiplexkonnektor in einer Sequenz an die Kompositionskomponente vermittelt (Eingangsport $P_{A3.1}$). Diese speichert zunächst alle ankommenden Datenobjekte. Die Zuordnung erfolgt über die Annotationsinformationen, die den Identifikator der Ursprungsnachricht sowie die Positionskennung des jeweiligen Datenobjektes in der Nachricht enthält. Hat die Kompositionskomponente alle Teildaten einer Nachricht erhalten, fügt sie diese zu einer zusammengesetzten Nachricht im MIME-Format zusammen und gibt diese am Port $P_{A3.1}$ aus. Die Kompositionskomponente stellt somit für die Nachrichtenteile ein Synchronisationspunkt dar, an dem erst alle Teile eintreffen müssen, bis die Nachricht weitergeleitet wird. Beide Komponenten können auch mehrere Nachrichten parallel verarbeiten. Ein verändertes Verhalten der Kompositionskomponente könnte auch ein progressives Anzeigen von Nachrichten ermöglichen. Die Komponente müsste dann beim Eintreffen jedes Nachrichtenteils eine Nachricht erzeugen und weitersenden. Unterstützt der Client die Aktualisierung von Nachrichten, kann er die Darstellung Nachricht Schritt für Schritt vervollständigen.

Der Teilgraph in Abbildung 5-12 adaptiert also selbst noch keine Nachrichten, zerlegt diese aber in die einzelnen Bestandteile und fügt die Ursprungsnachrichten auch wieder zusammen. Damit wird die Grundlage für eine fein-granulare Adaption von Nachrichtenbestandteilen gelegt.

5.2.3 Ersetzen von Daten

Im folgenden Beispiel werden E-Mail-Nachrichten durch die Ersetzung von Anhängen adaptiert (siehe Abbildung 5-13). Dazu wurde der Graph aus Abschnitt 5.2.2 um einige

Komponenten erweitert. Die Bestandteile der zerlegten Nachrichten werden dabei unterschiedlich behandelt. Die Kopfinformationen sowie der Nachrichtentext werden unverändert über die Ports $P_{A1.1}$ und $P_{A1.2}$ an die Kompositionskomponente weitergeleitet. Die Anhänge werden über den Port $P_{A1.3}$ ausgegeben und danach zunächst durch die Komponente „Schwellwertfilter“ verarbeitet. Dieser filtert eintreffende Datenobjekte unabhängig von ihrem Typ entsprechend der Größe der enthaltenen Daten (in Byte). Datenobjekte, deren Größe unterhalb des konfigurierten Schwellwertes liegt, werden am Ausgabeport $P_{A4.1}$ ausgegeben und unverändert an die Kompositionskomponente weitergeleitet. Datenobjekte, deren Größe den Schwellwert überscheiten, werden über Port $P_{A4.2}$ an den Bildauswahlkonnektor übergeben. Dieser vermittelt Daten entsprechend ihres Typs an die Ports $P_{E6.1}$ bzw. $P_{E7.1}$. Im Beispiel werden Datenobjekte des Typs (image) über $P_{E6.1}$ an die Komponente zur Bildersetzung weitergeleitet. In dieser wird eine Textbeschreibung als alternative Repräsentation verwendet. Die Textbeschreibung muss mit den Bilddaten vorliegen und deshalb vom Erzeuger der Bilddaten geliefert werden. Der Ersetzungsmechanismus entspricht damit einer Auswahl alternativer Ersetzungsdaten, die explizit mit den Primärdaten verfügbar sind[13].Alle weiteren Typen werden über $P_{E7.1}$ an die Komponente „Extrahierung“ vermittelt. Diese ersetzt die Datenobjekte durch eine Textbeschreibung, die aus Informationen über das ersetzte Datenobjekt generiert[14] wird. Unter anderem enthält der generierte Text einen Identifikator, der den Anhang der Nachricht eindeutig identifiziert und ein späteres Nachladen dieses Anhangs ermöglicht. Der Identifikator entspricht den Kompositionsinformationen aus Abschnitt 5.2.2.

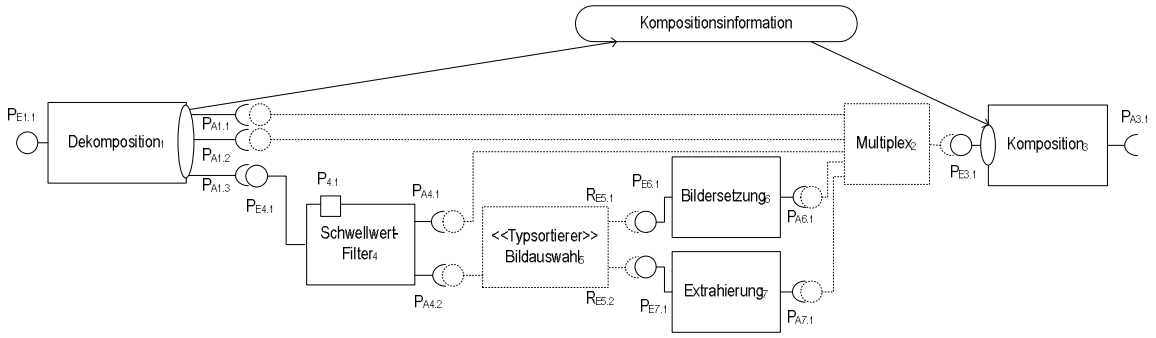


Abbildung 5-13: Teilgraph zur Ersetzung der Anhänge von E-Mail-Nachrichten

Die Ersetzung der Anhänge erfolgt transparent für die Komponenten „Dekomposition“ und „Komposition“. Die Steuerung der Ersetzung erfolgt über den Parameter $P_{4.1}$ der Schwellwertkomponente. Dieser ist von Typ Integer und definiert den Schwellwert für die Größe von Datenobjekten in Byte. Dieser Wert kann in Abhängigkeit von Informationen über die Ausführungsumgebung sowie von Benutzerinformationen gesetzt werden. Beispielsweise kann der Benutzer den Schwellwert auf 0 setzen, wenn er keine Anhänge übertragen möchte, einen festen Wert konfigurieren oder die Adaption dem System überlassen. Diese Einstellung wird durch einen Sensor S_1 repräsentiert. Dieser kann die Werte „0“ für „keine Anhänge“, einen positiven Wert als statischen Schwellwert sowie -1 für eine dynamische Ermittlung des Schwellwertes besitzen. Zur Ermittlung eines dynamischen Schwellwertes kann der Benutzer für jeden Anhang eine maximale Übertragungszeit definieren, aus der der Schwellwert berechnet werden soll (S_2). Als Wert zur Charakterisierung der Ausfüh-

[13] Der verwendete Ersetzungsmechanismus entspricht der Klasse Auswahl entsprechend der Klassifikation in Kapitel 3.

[14] Dies entspricht der Klasse Extrahierung entsprechend der Klassifikation in Kapitel 3.

rungsumgebung wird die aktuell verfügbare Datenrate des Zugangsnetzwerkes (S_3) verwendet.

S_1=(Benutzer.Anwendungseinstellung.Email.Schwellwert, Integer, “byte”, -1, null, passiv)

S_2=(Benutzer.Anwendungseinstellung.Email.Übertragungszeit, Integer, “s”, 2s, null, passiv)

S_3=(Verbindung.Zugangsnetzwerk.Sendekanal, Float, bit/s, null, null, passiv)

Die Abbildungsfunktion AF_1 greift auf diese Sensoren über ihre Parameter $P_{AF1.1}=S_1$, $P_{AF1.2}=S_2$ und $P_{AF1.3}=S_3$, zu. AF_1 kann dann wie folgt definiert werden:

$AF_1= P_{AF1.1}$ if ($P_{AF1.1}>=0$)

$AF_1= ((P_{AF1.3}/8)* P_{AF1.2})$ if ($(P_{AF1.1}=-1)$

Das Beispielszenario unterstützt damit eine benutzerdefinierte Datenreduzierung. Der Benutzer kann anhand zweier Werte eine Konfiguration für die Ersetzung erstellen, die durch eine Abbildungsfunktion und weitere Umgebungsinformationen auf einen konkreten Schwellwert umgesetzt werden. Die Berechnung des Schwellwert-Parameters erfolgt außerhalb und unabhängig von der Komponente „Schwellwertberechnung“. Die zugehörige Abbildungsfunktion kann explizit und separat von Entwickler definiert und unabhängig von der Komponentenimplementierung verändert werden.

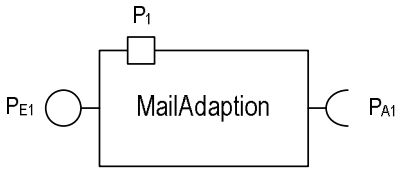


Abbildung 5-14: Zusammengefasste Komponente zur E-Mail-Adaption

In Abbildung 5-14 ist die zusammengesetzte Komponente zur Adaption von E-Mail-Nachrichten enthalten. Diese wird durch die Abbildung der Ports und Parameter von Komponenten aus Abbildung 5-13 definiert. Dafür werden der Port $P_{E1.1}$ auf P_{E1} und der Port $P_{A3.1}$ auf P_{A1} abgebildet. Außerdem wird der Parameter $P_{4.1}$ zur Definition der Schwellwerte für die einzelnen Objekttypen auf den Parameter P_1 abgebildet.

5.2.4 Adaption von Bilddaten

In diesem Abschnitt werden ein Graph zur Adaption der Bildgröße sowie des Bildformates und eine mögliche Parametrisierung der Komponenten zur Adaption von Bildern an die Displaygröße des Endgeräts beschrieben. Abbildung 5-15 stellt den zugehörigen Graph dar. Dieser besteht aus drei Komponenten. Die erste Komponente ($Rohformat_1$) ist parameterlos und besitzt einen Ein- und einen Ausgangsport. Die Signatur des Eingangsports $P_{E1.1}$ wird mit σ_1 bezeichnet und definiert alle Bilder der Formate GIF, JPEG, BMP, TIFF, PNG, FPX und PNM:

σ_1=set{(image/{gif|jpeg|bmp|tiff|png|fpx|pnm})}.

Die Komponente „Rohformat“ wandelt Bilder der angegebenen Formate in eine interne Darstellung um, die von Operationen der Java-basierten Programmierschnittstelle JAI [Sun99] verarbeitet werden kann. Die Signatur σ_2 des Ausgangsports $P_{A1.1}$ beschreibt dieses Bildformat:

$\sigma_2=\sigma_3=\sigma_4=\sigma_5$=set{(image/raw/jai)}

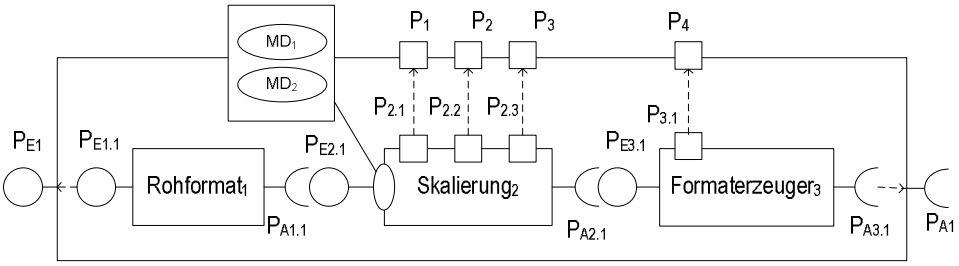


Abbildung 5-15: Beispielgraph zur Bildadaption

Die Komponente „Skalierung“ besitzt ebenfalls einen Ein- und einen Ausgangsport. Sie skaliert die räumliche Größe eines Bildes im Rohformat abhängig von den Parametern $P_{2.1}$, $P_{2.2}$ und $P_{2.3}$. Die Parameter $P_{2.1}$ und $P_{2.2}$ sind vom Typ Float und bestimmen die horizontale ($P_{2.1}$) und vertikale ($P_{2.2}$) Auflösung des Zielbildes. Diese werden als Skalierungsfaktoren mit einem Wertebereich zwischen 0 und 1 interpretiert. Der Parameter $P_{2.3}$ ist vom Typ String und bestimmt die zu verwendende Skalierungsmethode. In der Implementierung werden die Methoden „bicubic“, „bicubic2“ und „bilinear“ unterstützt. Die Signaturen des Ein- und Ausgabeports ($P_{E2.1}$ und $P_{A2.1}$) werden mit σ_3 und σ_4 bezeichnet und sind mit der Signatur σ_2 identisch.

$P_{2.1}$=(Float, „Skalierungsfaktor“, 1.0F, „0-1“)

$P_{2.2}$=(Float, „Skalierungsfaktor“, 1.0F, „0-1“)

$P_{2.3}$=(String, „Skalierungsmethode“, „bicubic“, „{bicubic, bicubic2, bilinear}“)

Die dritte Komponente des Graphen ist die Komponente „Formaterzeuger“. Diese wandelt Bilder nach der Adaption aus dem internen Format wieder in ein Bildformat um. Der Parameter $P_{3.1}$ legt das Zielformat fest. Der Parameter ist vom Typ String und kann die Werte GIF, JPEG, BMP, TIFF, PNG, FPX und PNM annehmen. Die Komponente besitzt ebenfalls einen Ein- und einen Ausgangsport. Die Signatur σ_5 ($P_{E3.1}$) entspricht der Signatur σ_4. Die Signatur des Ausgangsports entspricht den Formaten, in die das Bild konvertiert werden kann. Die von der Komponente erzeugbaren Formate entsprechen den durch den Parameter $P_{4.1}$ einstellbaren Formaten. Die Signatur des Ausgangsports σ_8 repräsentiert die unterstützten Formate:

$P_{4.1}$=(String, „Bildformat“, „JPEG“, „{GIF, JPEG, BMP, TIFF, PNG, FPX, PNM}“)

σ_6=set{(image/{gif|jpeg|bmp|wbmp|tiff|png|fpx|pnm})}.

Der in Abbildung 5-15 dargestellte Graph zur Adaption von Bilddaten kann in unterschiedlicher Weise parametrisiert werden, um Bilder an dynamische Eigenschaften der Ausführungsumgebung anzupassen. Wesentliche Eigenschaften der Ausführungsumgebung für Bilddaten sind die Displayeigenschaften und die Größe des Arbeitsspeichers des Endgeräts sowie die verfügbare Datenrate und weitere Eigenschaften der Kommunikationsverbindung zum Endgerät. In folgenden wird eine mögliche Parametrisierung der Komponenten zur Adaption von Bildern an die Displaygröße des Endgeräts sowie eine Anpassung an das Datenformat beschrieben. In die Betrachtungen werden die horizontale und vertikale Pixelanzahl des Displays einbezogen. Diese Informationen werden durch den Kontextdienst bereitgestellt und durch entsprechende Sensoren S_1 und S_2 beschrieben:

S_1=(Gerät.Display.x, Integer, “pixel”, 1280, “>0”, passiv)

S_2=(Gerät.Display.y, Integer, “pixel”, 1024, “>0”, passiv).

Diese Sensoren bilden die Grundlage zur Berechnung der Parameter $P_{2.1}$, $P_{2.2}$, $P_{2.3}$ und $P_{3.1}$. Außerdem werden Informationen über die anzupassenden Bilddaten benötigt. Dazu werden

entsprechende „MetaData“ Elemente definiert, die am Vorverarbeitungspunkt des Eingangsports $P_{E2.1}$ verfügbar sind. Für die Berechnung werden die horizontale und vertikale Auflösung des Originalbildes benötigt. Diese werden mit B.x und B.y definiert.

MD_1=(B.x, Integer, „pixel“, null, „>0“)

MD_2=(B.y, Integer, „pixel“, null, „>0“)

Die Abbildungsfunktionen AF_1 und AF_2 für die Parameter $P_{2.1}$ und $P_{2.2}$ besitzen jeweils zwei Parameter $P_{AF1.1}$ und $P_{AF1.2}$ bzw. $P_{AF2.1}$ und $P_{AF2.2}$, denen die Werte der Sensoren S_1 und S_2 bzw. der Metadaten MD_1 und MD_2 zugewiesen werden. Die Parameter $P_{2.1}$ und $P_{2.2}$ werden wie folgt definiert:

$P_{2.1}=AF_1=(P_{AF1.1}/ P_{AF1.2})$

$P_{2.2}=AF_2=(P_{A12.1}/ P_{AF2.2})$.

Der Parameter $P_{2.3}$ wird durch den Standardwert P_3=“bicubic“ definiert. Der Parameter $P_{3.1}$ wird durch das Standardbildformat des Endgerätes festgelegt. Dieses wird durch den Sensor S_3 bereitgestellt:

S_5=(Gerät. Medientyp.Bild.Standardformat, String, „Bildformat“, „gif“, null, passiv).

Der Parameterwert kann direkt durch den Sensorwert definiert werden:

$P_{3.2}=S_5$.

Dabei wird vorausgesetzt, dass das vom Endgerät erwartete Format auch von der Komponente „Formaterzeuger“ erzeugt werden kann.

Abbildung 5-15 definiert einen Graph zur Adaption des Formates sowie der räumlichen Größe von Bildern. Durch die Abbildung der Ports $P_{E1.1}$ auf P_{E1} und $P_{A3.1}$ auf P_{A1} sowie der Parameter $P_{2.1}$ auf P_1, $P_{2.2}$ auf P_2, $P_{2.3}$ auf P_3 und $P_{3.1}$ auf P_4 wird der Graph zu einer zusammengesetzten Komponente zusammengefasst. Diese wird als wiederverwendbarer Baustein in Abbildung 5-16 dargestellt. Die Signatur des Eingangsports wird mit $\sigma 1$, die Signatur des Ausgangsports durch σ_6 definiert.

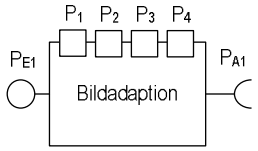


Abbildung 5-16: Komponente zur Adaption des Formates sowie der Größe von Bildern.

5.2.5 Kombination von Bildadaption und Bildersetzung

Die Komponente zur Bildadaption kann auf einfache Weise in das Beispiel aus Abschnitt 5.2.3 integriert werden. Dazu werden die Datenobjekte durch die Empfängerrolle $R_{E5.1}$ nicht direkt an die Komponente „Bildersetzung“ weitergeleitet sondern zunächst an einen Demultiplexer. Dieser vermittelt die Datenobjekte alternativ an die Komponenten „Bildadaption“ und „Bildersetzung“. Die Vermittlungsentscheidung kann z. B. anhand der Displaygröße (Bilder werden nur angezeigt, wenn das Display eine definierte Mindestgröße besitzt), Benutzereinstellungen („Bilder anzeigen“ oder „keine Bilder anzeigen“) oder Netzwerkeigenschaften (wenn verfügbare Datenrate > Schwellwert, dann „Bilder anzeigen“). Ein entsprechender Teilgraph ist in Abbildung 5-17 dargestellt.

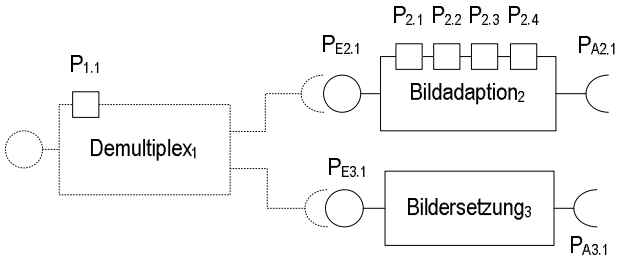


Abbildung 5-17: Teilgraph zur Kombination von Bildadaption und Bildersetzung

5.2.6 Integration von Kommunikations- und Datenadaption

Die Komponenten aus Abbildung 5-10, Abbildung 5-11 und Abbildung 5-14 werden in diesem Beispiel zu einer Gesamtanwendung zur Adaption von E-Mail-Nachrichten sowie deren Übertragung integriert (siehe Abbildung 5-18). Der MailClient kann Nachrichten über den Ausgangsport $P_{A5.1}$ anfordern. Diese werden in der Komponente „abgekoppelte Operation“ gemeinsam mit Anforderungen zum Vorabladen prioritätsgesteuert übertragen. Anforderungen des MailClients werden entweder aus dem Cache der abgekoppelten Operation beantwortet oder über den Ausgabeport $P_{A4.3}$ und den Multiplexer an die Komponente „Mailsource“ vermittelt. Erzeugte Anforderungen zum Vorabladen werden über den Ausgagnsport $A_{4.2}$ ausgegeben und ebenfalls über den Multiplexer an Eingangsport $E_{1.1}$ der Komponente „MailSource“ übergeben. Dafür gilt die Annahme, das die Komponente „MailSource“ sowohl Anfragen von konkreten Nachrichten über einen eindeutigen Bezeichner, als auch die Anfrage von Nachrichten anhand verschiedener Beschreibungsmerkmale unterstützt (z. B. Werte des Nachrichtenkopfes). Die Nachrichten werden über Port $P_{A1.1}$ an die Komponente „HeaderFilter“ übergeben, die die Nachrichten entsprechend ihrer Filterregeln (konfiguriert über Paramter $P_{2.1}$) zur Weiterleitung auswählt. Danach erfolgt in der Komponente „MailAdaption“ eine Folge von Adaptionsschritten, die auf die Anhänge der Nachrichten angewendet wird. Die Nachrichten werden dafür zunächst in ihre Bestandteile zerlegt. Diese werden dann einzeln adaptiert und abschließend wieder zu einer Nachricht zusammengefasst. Die adaptierten Nachrichten werden in der Komponente „Abgekoppelte Operation“ im Cache zwischengespeichert und an die Komponente „MailClient“ vermittelt. Außerdem kann die Komponente „MailSource“ proaktiv neue Nachrichten versenden, die ebenfalls durch die Adaptionskomponenten übertragen und angepasst werden. Die Konfiguration der Komponenten kann wie in den einzelnen Beispielen beschrieben erfolgen. Insbesonder ist in dem Gesamtszenario die Koordination der Parameterermittlung aus verschiedenen Kontextwerten möglich.

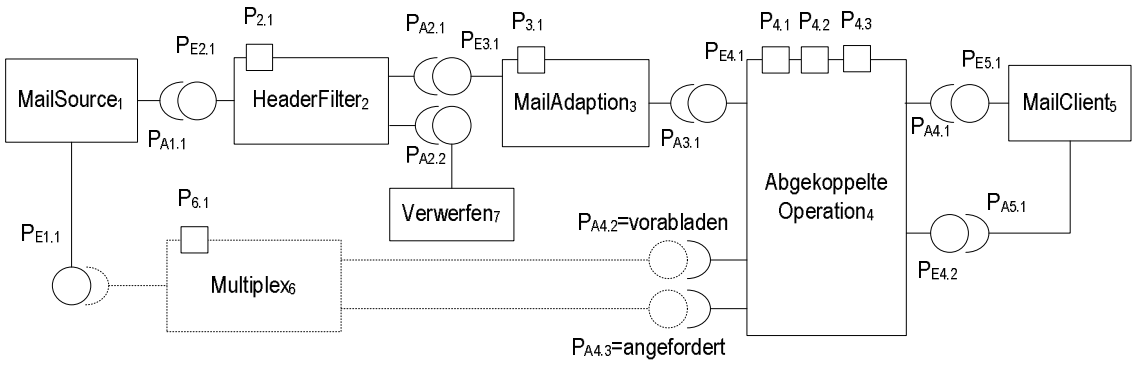


Abbildung 5-18: Integration von E-Mail-Adaption und abgekoppeltem Arbeiten

5.3 Adaption der Anwendungsstruktur

Die Adaption der Anwendungsstruktur kann durch alternative Pfade im Adaptionsgraphen modelliert werden. Die Kriterien zur Auswahl eines Pfades beschreiben dabei die Bedingungen zur Rekonfiguration der Anwendung zur Laufzeit. Der Entwickler kann somit auf der Ebene des Anwendungsmodells Bedingungen für eine Strukturadaption zur Laufzeit definieren und damit die Mechanismen zur Strukturadaption in die Entwicklung einbeziehen. Dies soll an einem einfachen Beispiel verdeutlicht werden. Abbildung 5-19 stellt einen Adaptionsgraph dar, der die Steuerung eines Servers über eine Spracheingabe beim Client ermöglicht. Die Sprachdaten werden in der Komponente „Spracheingabe“ erzeugt und über den Ausgangsport $P_{A1.1}$ an den Demultiplexer weitergeleitet. Dieser wählt entsprechend seines Parameters einen der beiden alternativen Pfade zur Weiterleitung der Sprachdaten aus. Die Verarbeitung der Sprache durch die „Sprache-in-Text“-Komponente reduziert die zu übertragende Datenmenge erheblich, da die Audiodaten durch Textdaten ersetzt werden. Als Entscheidungskriterium kann die Größe des Arbeitsspeichers sowie die verfügbare Rechenleistung des Endgerätes verwendet werden. Bleibt mindestens einer der Werte unterhalb des dafür definierten Schwellwertes, ist eine lokale Verarbeitung nicht möglich und die Sprachverarbeitung muss auf dem Server erfolgen. Stehen dagegen ausreichend Ressourcen zur Verfügung, kann der Client die Sprachdaten vor der Übertragung lokal verarbeiten.

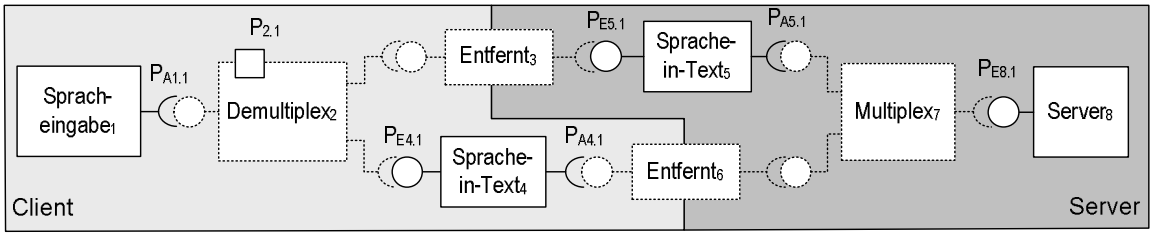


Abbildung 5-19: Alternative Platzierung einer Komponente zur Sprachverarbeitung

Das Beispiel setzt für den oberen Pfad eine Platzierung der „Sprache-in-Text“ Komponente auf dem Server, für den unteren Pfad auf dem Client voraus. Dies wird durch die Reihenfolge des Sequenzkonnektors „Entfernt“ in Verbindung mit der „Sprache-in-Text“ Komponente ausgedrückt. Unterstützt die Laufzeitplattform beispielsweise die dynamische Installation von Komponenten, kann das Entscheidungskriterium beim Start der Anwendung überprüft und die Komponenten entsprechend des gewählten Pfades platziert werden.

In ähnlicher Weise können Mechanismen zur Adaption der Struktur auf der Ebene der Komponenten modelliert werden. Für jede Konfiguration ist dabei die Beschreibung eines alternativen Pfades sowie entsprechender Entscheidungskriterien notwendig. Das Modell ermöglicht bzw. fordert dafür vom Entwickler die explizite Formulierung der Kriterien zur Auswahl einer bestimmten Anwendungsstruktur.

5.4 Bewertung

Nachfolgend soll das Meta-Modell hinsichtlich der Flexibilität bei der Beschreibung adaptiver Anwendungen für mobile und ubiquitäre Infrastrukturen bewertet werden. Den Ausgangspunkt dazu stellen die Vorbetrachtungen in Kapitel 1 dar. Dort werden die wesentlichen Anforderungen unter A1.1 bis A1.4 herausgearbeitet. Nachfolgend wird geprüft, welchen der Anforderungen die Validierungsbeispiele genügen.

Adaption der Datenübertragung

Im ersten Beispiel wird der Einsatz alternativer Kommunikationsprotokolle modelliert. Dabei werden Konnektoren verwendet, die an bestimmte Kommunikationsmedien bzw. Qualitätsparameter von Netzwerkverbindungen angepasst wurden. Dabei können beliebige Protokolle kombiniert werden. Im Beispiel wurden die Kompression von Paketköpfen zur Unterstützung asymmetrischer Verbindungen und I-TCP in Kombination mit SDP zur Datenübertragung über Verbindungen mit hoher Fehlerrate und häufigen gebündelten Fehlern verwendet. Durch die Verwendung alternativer Protokolle können die spezifischen Eigenschaften von Netzwerktechnologien unterstützt werden. Auswahlkriterien können durch die Kombination beliebiger Kontextwerte formuliert werden. Im Beispiel wurden Informationen über die Verbindung sowie über den Aufenthaltsort des Benutzers verwendet.

Eine Adaption an häufige Verbindungsunterbrechungen kann durch Queuing sowie durch die Steuerung der Übertragung erreicht werden. Die zusammengesetzte Komponente in Beispiel zwei unterstützt dieses Szenario. Außerdem wird die im ersten Beispiel definierte Komponente integriert. Queuing wird zur Zwischenspeicherung von Datenobjekten sowie im Sinne einer Verzögerung der Übertragung während Verbindungsunterbrechungen eingesetzt. Die Übertragungskomponente unterstützt die Übertragungswiederholung eines Datenobjektes nach einer Verbindungsunterbrechung. Weitere Datenobjekte werden in einer Warteschlange zwischengespeichert. In Anlehnung an die Q-RPC-Lösung in Rover kann dadurch eine zuverlässige und asynchrone Kommunikation über RPC Aufrufe realisiert werden. Die Komponenten unterstützen aber die Übertragung von Datenobjekten beliebigen Objekttyps.

Durch das Hinzufügen weiterer Warteschlangenkomponenten sowie einer Prioritätssteuerung würde das zweite Beispiel zu einer Prioritätswarteschlange erweitert. Eintreffenden Datenobjekten wird abhängig vom Eingangsport eine Priorität zugewiesen, die festlegt, in welcher Reihenfolge die Datenobjekte aus der Warteschlange entnommen werden. Diese kann z. B. in Kombination mit abgekoppelten Operationen eingesetzt werden, um Datenanforderungen des Caches gegenüber vorab zu ladenden bzw. zurück zu schreibenden Daten bevorzugt zu übertragen. Ebenso kann die zusammengesetzte Komponente verwendet werden, um die Datenübertragung konkurrierender Anwendungen eines Endgerätes zu koordinieren. Dabei kann beispielsweise der Benutzer entscheiden, welche Anwendung priorisiert wird. Außerdem können Informationen über die Aktivität des Benutzers einbezogen werden.

Das vierte Beispiel kombiniert die beschriebenen Lösungen zu einer zusammengesetzten Komponente, die ein abgekoppeltes Arbeiten mit Daten unterstützt, auf die nur lesend zugegriffen wird. Durch Caching und Vorabladen wird gesichert, dass benötigte Daten bei einer Abkopplung lokal auf dem Endgerät vorliegen. Welche Daten benötigt werden, kann entweder explizit durch den Benutzer definiert oder muss vorhergesagt werden. Zur Vorhersage können die Aktionen des Benutzers beobachtet sowie die Aktivität des Benutzers, sein Aufenthaltsort und weiterer Kontext verwendet werden. Durch die Kombination mit einer prioritätsgesteuerten Datenübertragung kann eine effizientere Ausnutzung der verfügbaren Datenrate, insbesondere bei Verbindungen mit einer geringen Datenrate erreicht werden. Die höchste Priorität erhalten im Beispiel von der Anwendung direkt gesendete Anforderungen von Daten. Vorab gesendete Anforderungen werden diesen untergeordnet.

Adaption von Anwendungsdaten

Die Filterung, d. h. die Auswahl sowie das Verwerfen der ausgewählten Daten, wird im ersten Beispiel zur Datenadaption modelliert. Diesem liegt das Szenario einer E-Mail-Anwendung zugrunde. Die Filterkomponente verwirft anhand der definierten Filterregeln vollständige Nachrichten. Die Filterregeln können direkt durch den Benutzer definiert bzw. aus dessen Aktivität abgeleitet werden. Beispielsweise können durch die Aktivität Regeln definiert werden, durch die nur Nachrichten zum Benutzer gesendet werden, die das Arbeitsumfeld betreffen. Ebenso kann durch entsprechende Regeln ein Spam-Filter definiert werden. Benutzerabhängige Definitionen können außerdem mit Informationen über das Endgerät und die verfügbare Netzwerkverbindung kombiniert werden, um beispielsweise die Anwendung einer von verschieden-restriktiven Filterdefinitionen zu koordinieren. Der verfügbare Speicher auf dem Endgerät spielt vor allem für die lokale Verfügbarkeit von Nachrichten eine Rolle. Dazu können durch die Filterung gezielt Nachrichten ausgewählt werden. In das Beispiel wurde die Komponente zur Adaption an Verbindungsunterbrechungen als Mechanismus zur Adaption der Kommunikation integriert.

Das Zerlegen und Zusammensetzen strukturierter Daten ermöglicht eine Restrukturierung der Daten sowie die Adaption von Nachrichtenteilen. Im zweiten Bespiel wird dies angewendet, um Anhänge von E-Mail-Nachrichten gezielt zu ersetzen. Daten werden dafür zunächst anhand ihrer Größe ausgewählt. Datenobjekte, deren Größe einen bestimmten Schwellwert überschreiten, werden ersetzt. Datenobjekte mit einer Größe unterhalb des Schwellwertes werden unverändert übertragen. Die Ersetzung erfolgt getrennt für Bilddaten sowie alle weiteren Datentypen. Bilddaten werden durch eine alternative Textbeschreibung ersetzt die mit den Bilddaten assoziiert werden müssen. Alle weiteren Anhänge werden durch einen Standardtext ersetzt, der aus den Datenobjekten erzeugt wird. Damit wird eine Reduzierung der Datengröße aber auch eine Ersetzung nicht unterstützter Medienformate sowie weiterer Geräteeigenschaften unterstützt.

Die zusammengesetzte Komponente zur Bildadaption dient zur Formattransformation sowie zur Skalierung von Bildern. Dadurch wir eine Änderung des Datenformates sowie der räumlichen Größe von Bildern erreicht. Eine Anpassung kann an die Displaygröße, die unterstützten Medienformate und den verfügbaren Speicher eines Endgerätes sowie an geringe Datenraten erfolgen. Diese Komponente wird in die Adaption der E-Mail-Nachrichten integriert, um Bilder entsprechend Benutzervorgaben, der Displaygröße und Netzwerkeigenschaften entweder zu skalieren oder zu ersetzen.

Adaption der Anwendungsstruktur

Mechanismen zur Strukturadaption sind Bestandteil fast aller Beispiele. Das abschließende Beispiel zur Adaption der Anwendungsstruktur veranschaulicht das wesentliche Vorgehen bei der Modellierung von Basismechanismen zur Strukturadaption. Diese werden in Form alternativer Pfade beschrieben. Das Kriterium zur Auswahl eines Pfades wird durch einen 1:n-Konnektor oder eine Komponente mit mehreren Ausgangsports modelliert. Durch diese Elemente kann der Entwickler die Änderung der Platzierung sowie das Hinzufügen, Entfernen und die Replikation von Komponenten explizit beschreiben und die Bedingungen definieren, zu denen die Strukturadaption stattfinden soll. Unterstützt das Laufzeitsystem eine dynamische Änderung der Platzierung bzw. eine Rekonfiguration von Komponentenstrukturen, können die alternativen Pfade direkt auf diese Mechanismen umgesetzt werden. Andernfalls können die Mechanismen zur Laufzeit ebenfalls durch alternative Pfade abgebildet werden, die beim Start der Anwendung vollständig instantiiert werden.

5.4.2 Zusammenfassung

Eine zusammenfassende Darstellung der Bewertung der Validierungsszenarien zur Kommunikations- und Datenadaption ist in Tabelle 5-1 enthalten.Mechanismen zur Strukturadaption sind Bestandteil aller Szenarien. Der Überblick zeigt, dass für jede Anforderung mindestens ein Validierungsbeispiel existiert, das der Anforderung genügt. Durch die schrittweise Integration einfacher Komponenten und Konnektoren zu komplexen zusammengesetzen Komponenten und Konnektoren wird auch die Unterstützung der Anforderungen in eine Anwendung integriert. Dies wird vor allem durch das Beispiel zur Integration der Daten- und Kommunikationsadaption in Abschnitt 5.2.6 verdeutlicht. In diesem wurden Basismechanismen aus dem überwiegenden Teil der Klassen des Klassifikationsschemas eingesetzt.

Validierungsbeispiel	**A1.1**	**A1.2**	**A1.3**	**A1.4**
alternative Kommunikationsprotokolle		hohe Fehlerrate, gebündelte Fehler, asymmetrische Verbindungen	Aufenthaltsort	
Adaption an Verbindungs-unterbrechungen				Verbindungs-unterbrechungen
prioritätsgesteuerte Übertragung		geringe Datenrate	Prioritätsvergabe durch den Nutzer, Aktivität	Verbindungs-unterbrechungen, Abkopplungen
abgekoppelte Operationen		hohe Verzögerung	Benutzerdefiniertes Vorabladen, Aktivität	Verbindungs-unterbrechungen, Abkopplungen
Filterung von Daten	Speicher	Datenrate, Übertragungszeit einer Nachricht	benutzerdefinierte Filterregeln, Aktivität	
Ersetzen von Daten	unterstützte Medienformate, kleines Display, geringe Rechenleistung	Datenrate, Übertragungszeit einer Nachricht	benutzerdefinierter Schwellwert	
Adaption von Bilddaten	Displaygröße, unterstützte Medienformate, Speicher	Datenrate, Übertragungszeit einer Nachricht	benutzerdefiniertes Ersetzen von Bildern	

Tabelle 5-1: Bewertung der Validierungsszenarien

Durch die Validierungsbeispiele konnte nachgewiesen werden, dass die identifizierten Basismechanismen Lösungsmöglichkeiten für die Problemstellungen bzw. Anforderungen mobiler verteilter Infrastrukturen darstellen. Außerdem wurde gezeigt, dass die Basismechanismen flexibel miteinander kombiniert werden können. Damit vergrößert sich auch der unterstützte Anforderungsbereich. So lassen sich Basismechanismen zur Datenadaption und Adaption der Kommunikation sowohl untereinander als auch miteinander kombinieren. Resultierende Adaptionsgraphen unterstützen eine Adaption der Anwendungsdaten und der Übertragung dieser.

Ebenso erfüllt das Meta-Modell die Anforderungen aus Kapitel 4. Die Validierungsbeispiele belegen die Möglichkeit der Integration von Struktur- und Parameteradaption zu Komponenten und Adaptionsgraphen, die in Form der Komponenten bzw. der definierten Struktur wiederverwendet werden können (A4.1 und A4.2). Die Modellelemente Sensor und Abbildungsfunktion ermöglichen die explizite Betrachtung der verwendeten Kontext-

werte. Komponenten beschreiben anhand der Struktur die eingesetzten Basismechanismen. Diese können durch Abbildungsfunktionen explizit auf Paramter abgebildet werden (Anforderung A4.3). Die Abbildungsfunktionen integrieren insbesondere die Betrachtung der Adaption an Endgeräte, Kommunikationsverbindungen sowie an Benutzerwünsche und die Anwendungssituation. Diese Informationen werden zu Parametern verknüpft und steuern damit die Adaption im Zusammenhang. Die Plattformunabhängigkeit des Meta-Modells wird durch die Modellierungen von Anwendungen auf der Strukturebene gesichert (A4.4).

Wesentliche Vorteile des Meta-Modells sind die flexible Kombination von Komponenten und Konnektoren durch ein einfaches Kommunikationsprinzip zur Vereinheitlichung von Signalisierung und Datenfluss, das durch die Konzepte der Datenobjekte sowie der Ports und Rollen erreicht wird. Dadurch können Nachrichten und multimediale Daten durch Kommponenten und Konnektoren einheitlich verarbeitet werden. Damit kann sowohl die Kommunikation zur Anforderung von Daten als auch die Übertragung von Daten adaptiert werden. Eine integrierte Betrachtung technischer und benutzerabhängiger Informationen in Form von Kontext ermöglicht die Koordination von Adaptionsprozessen. Die Abstraktion von Laufzeitumgebungen vereinfacht das Meta-Modell. Adaptionsgraphen können dadurch insbesondere auf eine Vielzahl von Plattformen umgesetzt werden, was für heterogene Infrastrukturen von besonderer Bedeutung ist.

Kapitel 6

ZUSAMMENFASSUNG UND AUSBLICK

6.1 Zusammenfassung

Die vorliegende Arbeit untersuchte wesentliche Mechanismen zur Adaption sowie den Überschneidungsbereich von Adaption und „Context-Awareness“. Ziel der Untersuchungen war es, gewonnene Erkenntnisse über Basismechanismen und Grundprinzipien der Adaption zur systematischen Entwicklung adaptiver Anwendungen zu benutzen.

Adaption wird im Rahmen der Arbeit vor allem als Fähigkeit der dynamischen Anpassung von Anwendungen verstanden. Diese Fähigkeit ist eine wesentliche Voraussetzung, um Anwendungen in heterogenen und dynamischen Infrastrukturen wie dem zukünftigen mobilen Internet verfügbar zu machen, die als wesentliche Zielinfrastrukturen für die Lösungsansätze der vorliegenden Arbeit betrachtet werden. Außerdem kann bestehenden Anwendungen durch die Fähigkeit der Adaption eine größere Ausführungsplattform erschlossen werden. Weiterhin ermöglicht sie im Vergleich zu einer manuellen Erstellung angepasster Anwendungsversionen eine wesentlich kostengünstigere und effizientere Anwendungsentwicklung.

Folgende Thesen wurden formuliert:

These 1: Es existieren wiederverwendbare und klassifizierbare Basismechanismen zur Adaption von Anwendungen an die Heterogenität und Dynamik mobiler verteilter Infrastrukturen.

These 2: Adaptive Anwendungen können durch die Kombination von Basismechanismen systematisch entwickelt und realisiert werden.

These 3: Informationen über die Ausführungsumgebung und die Anwendungssituation sowie Benutzereinstellungen können in Form von Kontext zur Steuerung der Adaption verwendet werden.

Identifizierung und Klassifizierung wiederverwendbarer Basismechanismen

Begonnen wurde mit einer Analyse des State-of-the-Art im Forschungsbereich Mobile und Ubiquitous Computing, um grundlegende Prinzipien und Mechanismen der Adaption zu erarbeiten. Ein erstes wesentliches Ergebnis der Arbeit ist die Identifizierung und Klassifikation wiederverwendbarer Basismechanismen zur Adaption. Von Bedeutung sind zum einen adaptive Mechanismen zur Verarbeitung und Übertragung von Anwendungsdaten, die in der Regel durch Codemodule realisiert werden, zum anderen die adaptive Auswahl, Kombination, und Platzierung dieser Codemodule. Grundlegend können Basismechanismen zur Struktur- und Parameteradaption unterschieden werden. Klassen der Parameter-

adaption sind Mechanismen zur Adaption von Anwendungsdaten sowie für ein angepasstes Kommunikationsverhalten. Die in Kapitel 3 vorgestellte Klassifikation von Basismechanismen zur Adaption belegt These 1.

Die Untersuchungen des Forschungsbereiches Context-Awareness zielten auf die Fragestellung woran angepasst wird. Diese kann durch Kontext beantwortet werden, der Informationen über die Ausführungsumgebung, Benutzer sowie die jeweilige Anwendungssituation repräsentiert. Kontext wird in Form eines Kontextdienstes für Anwendungen verfügbar. Wesentlich ist, dass Kontext die notwendigen Informationen zur Steuerung der Adaption liefert. Meta-Informationen, die in der Regel durch den Anwendungsentwickler bzw. Benutzer bereitgestellt werden, stellen wichtige Zusatzinformationen dar, die vor allem automatische Adaptionsprozesse steuern und deren Ergebnisse wesentlich verbessern können.

Ein Meta-Modell kontextabhängiger Adaptionsgraphen

Die identifizierten Basismechanismen sowie die Erkenntnisse über die Steuerung von Adaptionsprozessen wurden bei der Erarbeitung des Meta-Modells zur Beschreibung adaptiver Anwendungen aus der Sicht der Adaption angewendet. Elemente des Meta-Modells dienen der Beschreibung von Anwendungsmodellen. Das Meta-Modell stützt sich auf Kernelemente von Architekturbeschreibungssprachen. Zur Modellierung wurde demnach die Ebene der Architektur gewählt, die den Fokus der Betrachtungen auf die Struktur von Anwendungen legt. Auf dieser Ebene können sowohl die verwendeten Basismechanismen als auch deren Verbindung und Platzierung in integrierter Form beschrieben werden. Softwarekomponenten stellen die Basis zur Beschreibung der Adaptionsmechanismen dar. Dies unterstützt vor allem die Ersetzbarkeit und Wiederverwendbarkeit von Basismechanismen. Die Definition des Meta-Modells erfolgte auf der Basis der Meta-Object Facility (MOF) Spezifikation der Object Management Group (OMG). In der Arbeit wurde vor allem die graphische Notation von MOF auf Basis der Unified Modelling Language (UML) eingesetzt.

Basismechanismen werden im Meta-Modell durch parametrisierbare Komponenten und Konnektoren repräsentiert. Diese können zu Adaptionsgraphen komponiert werden. Die Komposition erfolgt durch die Verknüpfung von Komponentenports und Konnektorenrollen. Komponenten können mehrere Eingabe- und Ausgabeports besitzen und ermöglichen damit die Erzeugung von Graphenstrukturen. Adaptionsgraphen repräsentieren adaptive Anwendungen und beschreiben sowohl die eingesetzten Basismechanismen als auch deren Beziehungen. Sie sind gerichtet und können Zyklen enthalten. Komponenten und Konnektoren stellen die Knoten, die Verknüpfungen zwischen Ports und Rollen die Kanten des Graphen dar. Das Meta-Modell ermöglicht außerdem die Beschreibung zusammengesetzter Komponenten und Konnektoren. Damit können Anwendungsbausteine aus Basismechanismen hierarchisch komponiert werden. Adaptionsmechanismen können somit auf verschiedenen Detaillierungsebenen betrachtet und entworfen werden.

Die Zugriffe auf Kontext, d. h. Informationen über die Ausführungsumgebung, werden durch abstrakte Sensoren modelliert. Diese repräsentieren jeweils einen Kontextwert und abstrahieren von dessen konkreter Ermittlung. Die Berechnung von Parameterwerten zur dynamischen Konfiguration von Komponenten und Konnektoren erfolgt durch Abbildungsfunktionen. Diese berechnen aus Werten verschiedener Sensoren Parameterwerte für Komponenten und Konnektoren. Die Berechnung kann auch in mehreren Stufen erfolgen, wobei Resultatwerte von Abbildungsfunktionen als Eingabewerte weiterer Abbildungsfunktionen verwendet werden können. Meta-Informationen zur Steuerung der Adaptions-

mechanismen in Komponenten werden durch Annotationen modelliert. Diese werden den Datenobjekten eindeutig zugeordnet, deren Adaption gesteuert werden soll. Sowohl die Abbildung von Kontext auf Parameter als auch die Zuweisung und Nutzung von Annotationen in Komponenten kann durch Modellelemente explizit beschrieben werden. Anwendungsdaten und Signalisierungsnachrichten werden im Meta-Modell einheitlich als Datenobjekte betrachtet. Damit können Adaptionsmechanismen sowohl Mediendaten als auch auf Aufrufe und Signalisierungsnachrichten auf Protokollebene verarbeiten.

Die Validierung des Meta-Modells erfolgte anhand verschiedener Anwendungsszenarien, die durch Adaptionsgraphen beschrieben wurden. Ziel war der Nachweis der flexiblen Einsetzbarkeit des Meta-Modells, der Wiederverwendbarkeit einzelner Mechanismen und Teilgraphen sowie die Kombinierbarkeit verschiedenster Basismechanismen durch das entwickelte Meta-Modell. Die Beispiele wurden zur Darstellung der genannten Ziele gewählt. Sie beruhten zum einen auf Ansätzen aus der Literatur, zum anderen auf eigenen Implementierungen. Damit wurden sowohl die Dokumentations- und Analysemöglichkeiten des Modells als auch die Eignung für den Entwurf adaptiver Anwendungen veranschaulicht. Begonnen wurde mit verschiedenen Szenarien zur Adaption der Datenübertragung auf der Basis existierender Lösungen. Diese wurden schrittweise zu größeren Szenarien komponiert. Weiterhin wurden Szenarien zur Daten- und Strukturadaption integriert, die zu großen Teilen auf eigenen Implementierungen beruhten. Dabei wurden zusammengesetzte Komponenten und Konnektoren entwickelt und auf verschiedene Weise miteinander kombiniert. Die Validierung bezog insbesondere ein breites Spektrum der identifizierten Basismechanismen ein, um nachzuweisen, dass Mechanismen aller Klassen durch das Meta-Modell beschrieben und miteinander kombiniert werden können. Damit wurde die zweite These der Arbeit belegt. Die Steuerung der Adaption durch Kontext und Meta-Informationen wurde in den Beispielszenarien anhand konkreter Werte veranschaulicht und belegt, wodurch die dritte These bestätigt wurde.

6.2 Ausblick

Die Ergebnisse der vorliegenden Arbeit ermöglichen die Modellierung adaptiver Anwendungen in Form von Adaptionsgraphen. Diese bieten vielfältige Anknüpfungspunkte für weitere Forschungsarbeiten. Eine wesentliche Frage bei der Entwicklung adaptiver Anwendungen ist der effiziente Einsatz von Adaptionsmechanismen. Dieser wird durch die Auswahl und Platzierung der Mechanismen sowie deren Kombination und Parametrisierung beeinflusst. Eine Bewertung von Adaptionsgraphen hinsichtlich der Adaptionsergebnisse ist vor allem für eine Auswahl von Lösungen aus mehreren alternativen Graphen oder Pfaden von Bedeutung. Lösungsansätze hierzu enthält [CRS98] mit der Beschreibung von Algorithmen zur Erzeugung optimaler Sequenzen von Medienfiltern.

Das Meta-Modell beschreibt adaptive Anwendungen unabhängig von Komponentenplattformen. Wesentliche Problemstellungen in diesem Zusammenhang sind die Transformation von Adaptionsgraphen in lauffähigen Code sowie die Entwicklung adaptiver Anwendungen auf Basis von Komponenten. Letzterer Punkt betont vor allem die Notwendigkeit einer softwaretechnischen Unterstützung der Adaption durch entsprechende Sprachkonzepte, Entwicklungsmethoden und Werkzeuge.

Die Mechanismen zur Strukturadaption werden durch die bedingte Vermittlung von Datenobjekten über alternative Pfade modelliert. Bedingungen können durch die Parametrisierung der Verzweigungselemente formuliert werden. Auf diese Weise kann eine Änderung der Platzierung sowie eine Rekonfiguration zur Laufzeit beschrieben werden. Die

Möglichkeiten bleiben aber auf die Definition statischer Graphen beschränkt. Erweiterungsmöglichkeiten bestehen somit für die Lösung durch die Definition von Elementen zur expliziten Beschreibung von Strukturänderungen bzw. zur dynamischen Erzeugung von Strukturen. Ausgangspunkte für weiterführende Überlegungen bieten die Konzepte in [MDK94]. Die Platzierung von Komponenten wird in der Lösung im Rahmen der Arbeit ebenfalls nur implizit durch die Definition von Konnektoren zur entfernten Kommunikation beschrieben. Eine explizite Beschreibung der Ausführungsorte und darauf aufbauend eine automatische Platzierung von Graphenelementen könnten anhand der Ansätze in [KRW+01] untersucht werden.

Basis für weitere Untersuchungen bietet auch die Problemstellung der Definition von Abbildungsfunktionen von Kontext auf Steuerinformationen. Diese können durch das Meta-Modell nur beschrieben werden. Die Definition der Abbildungsfunktionen bleibt Aufgabe des Entwicklers. Abbildungsfunktionen sind in der Regel spezifisch für bestimmte Anwendungen bzw. Zustände der Ausführungsumgebungen. Die Untersuchung von Standardfunktionen und –konfigurationen für bestimmte Anwendungsfälle und die Definition von Abbildungsfunktionen für spezifische Problemstellungen stellen ebenfalls wesentliche Herausforderungen für eine weitere Forschung dar.

LITERATURVERZEICHNIS

[AAH+97] Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R., Pinkerton, M.: *Cyberguide: A mobile context-aware tour guide.* Wireless Networks Vol. 3, No. 5, S. 421–433, 1997.

[AAH+98] Alexander, D. S., Arbaugh, W. A., Hicks, M. W., Kakkar, P., Keromytis, A. D., Moore, J. T., Gunter, C. A., Nettles, S. M., Smith, J. M.: *The SwitchWare Active Network Architecture.* IEEE Network, Special Issue on Active and Controllable Networks, 1998.

[ABG+97] Alexander, D.S., Braden, B., Gunter, C.A., Jackson, A.W., Keromytis, A.D., Minden, G.J., Wetherall, D.: *Active Network Encapsulation Protocol (ANEP).* RFC DRAFT, 1997.

[AlG97] Allenand, R., Garlan, D.: *A Formal Basis for Architectural Connection*. In: ACM Trans. in Software Engineering and Methodology, Vol. 6, No.3, S. 213-249, 1997.

[APL+95] Ayanoglu, E., Paul, S., LaPorta, T. F., Sabnani, K. K., Gitlin, R. D.: *AIR-MAIL: A Link-Layer Protocol for Wireless Networks.* ACM Wireless Networks, Vol. 1, No. 1, S. 47-60, 1995.

[APS99] Allman, M., Paxson, V., Stevens, W.: *TCP Congestion Control.* RFC2581, Network Working Group, 1999.

[BaB95a] Bakre, A., Badrinath, B. R.: *I-TCP: Indirect TCP for Mobile Hosts.* In: Proc. 15th International Conf. on Distributed Computing Systems (ICDCS), Mai 1995.

[BaB95b] Bakre, A., Badrinath, B.R.: *M-RPC: A Remote Procedure Call Service for Mobile Clients.* In: Proceedings of the 1st ACM Mobicom Conference, S. 2-11, 1995.

[BaK98] Balakrishnan, H., Katz, R.: *Explicit Loss Notification and Wireless Web Performance.* In: Proceedings of IEEE GLOBECOM, 1998.

[BaP01] Balakrishnan, H., Padmanabhan, V. N.: *How Network Asymmetry Affects TCP.* IEEE Communications Magazine, Vol. 39, No. 4, S. 60-67, 2001.

[BBI+93] Badrinath, B.R., Bakre, A., Imielinski, T., Marantz, R.: *Handling Mobile Clients: A Case for Indirect Interaction.* In: Proc. of the 4th Workshop on Workstation Operating Systems (WWOS-IV), 1993.

[Ber00] Berst, J.: *Finally, A Silicon Valley Buzzword Worth Remembering: Evernet.* www.zdnet.com, 2000.

[BGP01] Buyukkokten, O., Garcia-Molina, H., Paepcke, A.: *Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices.* Proc. of 10th Int. World Wide Web Conference. Hong Kong, S. 652-662, 2001.

[BGS99] Bickmore, T., Girgensohn, A., Sullivan, J. W.: *Web Page Filtering and Re-Authoring for Mobile Users.* In: The Computer Journal, Vol. 42, No. 6, S. 534-546, 1999.

[BIP+99] Badrinath, B. R., Imielinski, T., Phatak, S., Sudame, P., Deshpande, P.: *Opportunistic FTP.* Unveröffentlichtes internes Dokument, Rutgers Universität, New Jersey, DATAMAN Lab, 1999.

[BMR+96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: *Pattern-oriented Software Architecture – A System of Patterns Volume 1.* ISBN 0471958697, John Wiley and Sons Verlag, 1996.

[Bog73] Boggs, J.K.: *IBM Remote Job Entry Facility: Generalize Subsystem Remote Job Entry Facility.* IBM Technical Disclosure Bulletin 752, IBM. 1973.

[BPS+97a] Balakrishnan, H., Padmanabhan, V.N., Seshan, S., Katz, R.H.: *A Comparison of Mechanisms for Improving TCP Performance over Wireless Links.* IEEE/ACM Transactions on Networking, Vol. 5, No. 6, S. 756-769, 1997.

[BPS+97b] Balakrishnan, H., Padmanabhan, V. N., Seshan, S., Stemm, M., Amir, E., Katz, R. H.: *TCP Improvements for Heterogeneous Networks: The Daedalus Approach.* In: Proceedings of the 35th Annual Allerton Conference on Communication, Control, and Computing, 1997.

[Bra98] Braam, P. J.: *The Coda Distributed File System.* Linux Journal, No. 50, 1998.

[BrW98] Brown, A. W., Wallnau, K. C.: *The Current State of CBSE.* IEEE Software, Vol. 15, No. 5, S. 37-46, 1998.

[BSA+95] Balakrishnan, H., Seshan, S., Amir, E., Katz, R.H.: *Improving TCP/IP Performance over Wireless Networks.* In: Proceedings of Mobicom, November `95 In Proc. 1st ACM Int'l Conf. on Mobile Computing and Networking (Mobicom95), 1995.

[BSK95] Balakrishnan, H., Seshan, S., Katz, R. H.: *Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks.* ACM Wireless Networks, Vol. 1, No. 4, 1995.

[BZB+97] Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S.: *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification.* RFC2748, 1997.

[CaI95] Cáceres, R., Iftode, L.: *Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments.* IEEE Journal on Selected Areas in Communications, Vol. 13, No. 5, 1995.

[CBZ+98] Calvert, K.L., Bhattacharjee, S., Zegura, E.W., Sterbenz, J.P.G.: *Directions in Active Networks*. IEEE Communications, Vol.36 No. 10, S. 72–78, 1998.

[CDK94] Coulouris, G., Dollimore, J., Kindberg, T.: *Distributed Systems: Concepts and Design.* 2. Ausgabe, ISBN 0-201-62433-8, Addison-Wesley, 1994

[CDM+00] Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.: *Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences.* In: Proceedings of the CHI 2000 Conference on Human Factors in Computing Systems 2000, Den Haag, ACM Press, New York 2000, S. 17-24, 2000.

[ChD00] Cheesman, J., Daniels, J.: *UML Components – A Simple Process for Specifying Component-Based Software.* Addison-Wesley Verlag, ISBN 0201708515, 2000.

[CMI99] Chandrasekaran, S., Madden, S., Ionescu, M.: *Ninja Paths: An Architecture for Composing Services over Wide Area Networks.* CS262 class project writeup, ninja.cs.Berkeley.edu, 1999.

[CMK+99] Campbell, A.T., De Meer, H.G., Kounavis, M.E., Miki, K., Vicente, J.B., Villela, D.: *A Survey of Programmable Networks.* Computer Communication Review, vol. 29, no. 2, S. 7-23, Apr. 1999.

[CRS98] Candan, K.S., Rangan, P.V., Subrahmanian, V.S.: *Collaborative Multimedia Systems: Synthesis of Media Objects.* IEEE Transactions on Knowledge and Data Engineering. May/June 1998, Vol. 10, No. 3, S.433-457, 1998.

[CTW01] Choi, S., Turner, J., Wolf, T.: *Configuring Sessions in Programmable Networks.*, In: Proceedings of IEEE INFOCOM 2001, S. 60-66, 2001.

[Dey00] Dey, A. K.: *Providing Architectural Support for Building Context-Aware Applications.* PhD Thesis, College of Computing, Georgia Institute of Technology, December 2000.

[Dey01] Dey, A. K.: *Understanding and Using Context.* In: Personal and Ubiquitous Computing, Vol. 5, No. 1, S. 4-7., 2001.

[DiJ01a] Ding, W., Jamalipour, A.: *Delay Performance of the New Explicit Loss Notification TCP Technique for Wireless Networks.* 2001.

[DiJ01b] Ding, W., Jamalipour, A.: *A new explicit loss notification with Electricityment for wireless TCP.* In: Proceedings of Personal, Indoor and Mobile Radio Communications Symposium (PIMRC2001), 2001.

[DPS+94] Demers, A. J., Petersen, K., Spreitzer, M. J., Terry, D. B., Theimer, M. M. , Welch, B. B.: *The Bayou architecture: Support for data sharing among mobile users.* In: Proceedings IEEE Workshop on Mobile Computing Systems & Applications, S.2–7, 1994.

[EAK+01] Elrad, T., Aksit, M., Kiczales, G., Lieberherr, K., Ossher, H.: *Discussing Aspects of AOP.* Communications of the ACM, Vol. 44, No. 10, S. 33-38, 2001.

[ECD+01] Efstratiou, C., Cheverst, K., Davies, N., Friday, A.: *An Architecture for the Effective Support of Adaptive Context-Aware Applications.* In: Proceedings of 2nd International Conference in Mobile Data Management (MDM`01), Hong Kong, Springer, Lecture Notes in Computer Science Vol. 1987, S. 15-26, January, 2001.

[EdE99] Eddon, G; Eddon, H.: *Inside COM+, COM+ Architektur und Programmierung*. ISBN: 3860634984, Microsoft Press Deutschland, 1999.

[EFB01] Elrad, T., Filman, R. E., Bader, A.: *Aspect-Oriented Programming.* Communications of the ACM, Vol. 44, No. 10, S. 29-32, 2001.

[EFD+02a] Efstratiou, C., Friday, A., Davies, N., Cheverst, K.: *Utilising the event calculus for policy driven adaptation on mobile systems.* In: Proceedings of the 3 International Workshop on Policies for Distributed Systems and Networks (POLICY, IEEE Computer Society, S. 13-24, 2002.

[EFD+02b] Efstratiou, C., Friday, A., Davies, N., Cheverst, K.: *A Platform Supporting Coordinated Adaptation in Mobile Systems.* In: Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02), Callicoon, New York, U.S., IEEE Computer Society, S. 128-137, June, 2002.

[EMP+97] Edwards, W. K., Mynatt, E.D., Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M. M.: *Designing and Implementing Asynchronous Collaborative Applications with Bayou.* In: Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), S.119–128, 1997.

[ETS00] ETSI: *Radio Broadcasting System; Digital Audio Broadcasing (DAB) to mobile, portable and fixed receivers.* ETSI Standard EN 300 401 V1.3.2, 2000.

[FDB+99] Friday, A., Davies, N., Blair, G. S., Cheverst, K.: *Developing adaptive applications: The MOST experience.* Journal of Integrated Computer-Aided Engineering, Vol. 6, No. 2, S. 143-157, 1999.

[FGC+97] Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A., Gauthier, P.: *Cluster-Based Scaleable Network Services.* Proceedings of the 16 th ACM Symposium on Operating System Principles, 1997.

[FGC+98] Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A.: *Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives.* In: IEEE Personal Communications, Vol. , No. 8, S. 10–19, 1998.

[FGG+98] Fox, A., Goldberg, I., Gribble, S.D., Polito, A., Lee, D.C.*: Experience with Top Gun Wingman: A proxy-based graphical web browser for the Palm Pilot PDA.* IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98). Lake District, UK, S. 15-18. 1998.

[FiG94] Fiuczynski, M.E., Grove, D.: *A Programming Methodology for Disconnected Operation.* Technischer Bericht, Universität Washington. 1994.

[Fis03] Fischmeister, S.: *Mobile Software Agents for Location Based Systems.* In: Agent Technologies, Infrastructures, Tools, and Applications for E-Services: NODe 2002 Agent-Related Workshops, Erfurt, Germany, October 7-10, 2002. Revised Papers. LNCS 2592, S. 226-239, 2003.

[FlJ93] Floyd, S., Jacobson, V.: *Random Early Detection (RED) Gateways for Congestion Avoidance.* IEEE/ACM Transactions on Networking, Vol. 1, No. 4, S. 397-413, 1993.

[FMS+98] Feldmeier, D.C., McAuley, A. J., Smith, J. M., Bakin, D.S., Marcus, W. S., Raleigh, T. M.: *Protocol boosters.* IEEE Journal on Selected Areas in Communications (JSAC) Vol. 16, No. 3, S 437-444, 1998.

[FoB96] Fox, A., Brewer, E. A.: *Reducing WWW-Latency and Bandwidth Requirements by Real-Time Distillation.* In: Proceedings of the Fifth International World Wide Web Conference, Paris, 1996.

[FoZ94] Forman, G. H., Zahorjan, J.: *The Challenges of Mobile Computing.* IEEE Computer, Vol. 27, No. 4, S. 38-47, 1994.

[FPV98] Fuggetta, A., Picco, G.P., Vigna, G.: *Understanding Code Mobility.* IEEE Trans. On Software Engineering, Vol. 24, No. 5, 1998.

[FrB96] Freed, N., Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.* RFC 2046, 1996.

[FrB98] Froese, K. W., Bunt, R.B.: *Scheduling Reads and Writes for a Weakly Connected Mobile Environment.* In Proceedings of the 10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, 1998.

[Füh00] Führer, D.: *ADSL, High-Speed Multimedia per Telefon*, Hüthig Verlag, ISBN: 3826650131, 2000.

[GBH98] Greenberg, M. S., Byingtion, J. C., Harper, D. G.: *Mobile Agents and Security.* IEEE Communications Magazin Vol. 36, No. 7 (1998), S.76–85, 1998.

[GHJ+01] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Entwurfsmuster – Elemente wieder verwendbarer objekt-orientierter Software.* Addison Wesley, ISBN: 3827318629, 2001.

[Gil00] Gilder, G.: *The Twenty Laws of the Telecosm*. www.kurzweilai.net/articles/-art0004.html?printable=1, 2000

[GMW97] Garlan, D., Monroe, R., Wile, D.: *ACME: An Architecture Description Interchange Language.* Proceedings of CASCON '97, S.169-183, 1997.

[GoA03] Goulao, M., e Abreu, F. B.: *Bridging the Gap between ACME and UML 2.0 for CBD.* 2003.

[Gol66] Golomb, S. W.: *Run-length encoding.* IEEE Transactions on Information Theory, Vol. 12, No. 3, S. 399-401, 1966.

[Gos01] Goslar, K.: *Gewinnung, Darstellung und Verwaltung von Kontextinformationen – Personalisierung.* Diplomarbeit, Lehrstühle für Rechnernetze sowie Wirtschaftsinformatik, TU Dresden, 2001.

[Gre01] Greenfield, J.: *UML Profile for EJB.* Rational Software Corporation, Public Draft, 2001.

[GrF97] Gribble, S. D., Fox, A.: *The Case for TACC: Scalable Infrastructure Servers for Transformation, Customization, and Caching.* CS286 Term Project, University of California at Berkeley, 1997.

[GrS01] Gray, P., Salber, D.: *Modelling and Using Sensed Context Information in the Design of Interactive Applications.* In: Reed Little, M., Nigay, L. (Hrsg.), EHCI 2001, Lecture Notes in Computer Sciences, Volume 2254, Springer Verlag, Berlin, S. 317–335, 2001.

[GWB+01] Gribble, S. D., Welsh, M., von Behren, R., Brewer, E. A., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Josheph, A., Katz, R.H., Mao, Z. M., Ross, S., Zhao, B.: *The Ninja Architecture for Robust Internet-Scale Systems and Services.* Special Issue on Pervasive Computing, Vol. 35, No. 4, 2001.

[Haa97] Haas, Z.J.: *Mobile-TCP: An Asymmetric Transport Protocol Design for Mobile Systems.* IEEE International Conference on Communications (ICC'97), Montreal, Kanada, Vol. 2, S.1054-1058. 1997.

[HCK97] Harrison, C., Chess, D., Kershbaum, A.: *Mobile Agents: Are They a Good Idea?.* In: Vitek, J., Tschudin, C. (eds.), Mobile Object Systems: Towards the Programmable Internet, Lecture Notes in Computer Science., Vol. 1222, S. 25-27. Linz: Springer 1997

[HKO+00] Hori, M.; Kondoh, G.; Ono, K.; Hirose, S.; Singhal S.: *Annotation-Based Web Content Transcoding.* 9th International World Wide Web Conference (WWW9), Amsterdam, Niederlande, 15.-19. Mai 2000.

[HNS00] Hofmeister, C., Nord, R., Soni, D.: *Applied Software Architecture.* ISBN 0-201-32571-3, Addison Wessley Longman Verlag, 2000.

[Huf52] Huffman, D. A.: *A Method for the Construction of Minimum Redundancy Codes.* In: Proceedings of Institute of Electronics and Radio Engineers, Vol. 40, No. 9, S. 1098-1101, 1952.

[HuH93] Huston, L. B., Honeyman, P.: *Disconnected Operation for AFS.* In Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent ComI, Camebridge, 1993.

[HuH95] Huston, L.B., Honeyman, P.: *Partially Connected Operation.* Computing Systems. Vol. 8, No. 4, S. 365-379, 1995.

[IKV00] IKV++: *Grasshopper Basics and Concepts, Release 2.2.* IKV++, www.grasshopper.de, 2000.

[Jac88] Jackobson, V.: *Congestion Avoidance and Control.* In: Proc. ACM SIGCOMM 88, 1988.

[Jac90] Jacobson, V.: *Compressing TCP/IP Headers for Low-Speed Serial Links.* RFC 1144, 1990.

[JFW02] Johanson, B., Fox, A., Winograd, T.: *The Interactive Workspace Project: Experiences with Ubiquitous Computing Rooms.* IEEE Pervasive Computing, Vol. 1, No. 2., S. 71-78, 2002.

[JLH+88] Jul, E., Levy, H., Hutchinson, N., Black, A.: *Fine-grained Mobility in the Emerald System.* ACM Transactions in Computer Systems, Vol. 6, No. 2, S.109-133, 1988.

[JLT+95] Joseph, A. deLespinasse, A. Tauber, J. Gifford, D. Kaashoek, F.: *Rover: A Toolkit for Mobile Information Access.* In: Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), S. 156-171, 1995.

[Joh98] Johansen, D.: *Mobile Agent Applicability.* Lecture Notes in Computer Science 1477 Mobile Agents, Second International Workshop, MA'98, Springer, ISBN 3-540-64959-X, 1998.

[JoK96] Joseph, A. D., Kaashoek, M. F.: *Building Reliable Mobile-Aware Applications using the Rover Toolkit.* In: Second ACM International Conference on Mobile Computing and Networking (MobiCom'96), 1996.

[JöM99] Jörding, T., Michel, S.: Personalized Shopping in the Web by Monitoring the Customer. In: Proceedings of "The Active Web – A British HCI Group Day Conference" Stafford, UK, 1999.

[JTK97] Joseph, A. D., Tauber, J. A., Kaashoek, M. F: *Mobile Computing with the Rover Toolkit.* IEEE Transactions on Computers: Special issue on Mobile Computing, Vol. 46, No. 3, S. 337-352, 1997.

[Kad03] Kadner, K.: *Konzeption eines verteilten Dienstes zur Unterstützung von Context-Awareness.* Großer Beleg, Lehrstuhl Rechnernetze, TU Dresden, 2003.

[KeM96] Keshav, S., Morgan, S.: *Smart Retransmission: Performance with Overload and Random Losses.* www.cs.att.com/netlib/att/cs/home/keshav/-papers/smart.ps.Z, Preprint, 1996.

[KHH+01] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W. G.: *Getting Started with AspectJ.* Communications of the ACM, Vol. 44, No. 10, S. 59-65, 2001.

[KiF00] Kiciman, E., Fox, A.: *Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment.* In: Proceedings of Second International Symposium on Handheld and Ubiquitous Computing, Springer-Verlag, S. 211-226, 2000.

[KiS92] Kistler, J. J. and Satyanarayanan, M.: *Disconnected Operation in the Coda File System.* ACM Transaction on Computer Systems, Vol. 10, No. 1, Februar 1992.

[Kis96] Kistler, J. J.: *Disconnected Operation in a Distributed File System.* Lecture Notes in Computer Science 1002, Springer-Verlag Berlin, Heidelberg, New York, ISBN 3-540-60627-0, 1996.

[KMN89] Kramer, J., Magee, J., Ng., K.: *Graphical Configuration Programming.* IEEE Computer, Vol. 22, No. 10, S. 53-65, 1989.

[KMS+92] Kramer, J., Magee, J., Sloman, M., Dulay, N.: *Configuring object-based distributed programs in Rex.* IEEE Software Engineering Journal, Vol. 7, No. 2, S. 139-149, 1992.

[KoB98] Kozaczynski, W., Booch, G.: *Component-Based Software Engineering.* IEEE Software, Vol. 15, No. 5, S. 34-36, 1998.

[KRW+00] Keller, R., Ramamirtham, J., Wolf, T., Plattner, B.: *Programming Active Networks Using Active Pipes.* Technical Report WUCS-00-27, Department of Computer Science, Washington University in St. Louis, MO, 2000.

[KRW+01] Keller, R., Ramamirtham, J., Wolf, T., Plattner, B.: *Active Pipes: Service Composition for Programmable Networks.* Milcom, 2001.

[KuS93a] Kumar, P., Satyanarayanan, M.: *Log-Based Directory Resolution in the Coda File System.* In: Proceedings of 2nd Int'l. Conf. on Parallel and Distributed Info. Sys., San Diego, CA, 1993.

[KuS93b] Kumar, P., Satyanarayanan, M.: *Supporting Application-Specific Resolution in an Optimistically Replicated File System.* Proceedings of the Fourth IEEE Workshop on Workstation Operating Systems, S. 66-70, 1993.

[KuS95] Kumar, P., Satyanarayanan, M: *Flexible and Safe Resolution of File Conflicts.* In: Proceedings of 1995 USENIX Tech. Conf., New Orleans, LA, 1995.

[LaC96] Lange, D.B., Chang, D. T.: *IBM Aglets Workbench, Programming Mobile Agents in Java.* IBM Corporation, 1996.

[Lam78] Lamport, L.: *Time, clocks, and the ordering of events in a distributed system.* Communications of the ACM, Vol. 21, No. 7, S. 558-565, 1978.

[Lan84] Langdon, G.: *An Introduction to Arithmetic Coding.* IBM Journal of Research and Development, No. 38, S. 135-149, 1984.

[Lan89] Langenscheidt: *Langenscheidts Fremdwörterbuch*. Verlag: Langenscheidt, ISBN 3468203969, Erscheinungsjahr: 1989.

[LaO99] Lange, D. B., Oshima, M.: *Seven Good Reasons for Mobile Agents.* Communications of the ACM, Vol. 42, No. 3, S. 88-89, 1999.

[LiS00] Lieberman, H., Selker, T.: *Out of Context: Computer Systems That Adapt To, and Learn From, Context.* Media Laboratory, Massachusetts Institute of Technology, Cambridge (USA) 2000. http://lieber.www.media.mit.edu/-people/lieber/Teaching/Context/Out-of-Context-Paper/Out-of-Context.html, Abruf am 2002-03-19.

[LiS88] Liskov, B., Shira, L.: *Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems.* In: Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation (PLDI), SIGPLAN Notices Vol. 23, No. 7, S.260-267,1988.

[LJP93] Lea, R., Jacquemont, C., Pillevesse, E.: *COOL: System Support for Distributed Object-Oriented Programming.* Communications of the ACM, Vol. 36, No. 9, S. 37-46, 1993.

[Lös02] Löschau, F.: *Realisierung kontextsensitiver Anwendungen auf Basis eines Wiemeworks zur Kontextverarbeitung und –verwaltung.* Diplomarbeit, Lehrstühle für Rechnernetze sowie Wirtschaftsinformatik, TU Dresden, 2002.

[LuA94] Luotonen, A., Altis, K.: *World-Wide Web Proxies.* 1994.

[LuV95] Luckham, D.C., Vera, J.: *An Event-Based Architecture Definition Language.* IEEE Transactions in Software Engineering, Vol. 21, No. 9, S. 717-734, 1995.

[Man03] Bregni, S., Mancuso, M., Pattavina, A.: *A Novel Scheme of Loss Differentiation and Adaptive Segmentation to Enhance TCP Performance over Wireless Networks.* Submitted to ICC2003, Anchorage, Alaska, USA, May 2003.

[Mar00] Marder, U.: *Transformation Independence in Multimedia Database Systems.* SFB-Report 11/2000, SFB 501 University of Kaiserslautern, November 2000.

[Mar01a] Marder, U.: *Transformation Independence for Multimedia Systems.* Internal Report, University of Kaiserslautern, 2001.

[Mar01b] Marder, U.: *On Realizing Transformation Independence in Open, Distributed Multimedia Information Systems.* In: Heuer, A., Leymann, F., Priebe, D. (eds.), Proc. Of 9. GI-Fachtagung ‚Datenbanksysteme in Büro, Technik und Wissenschaft', BTW ‚2001, Springer-Verlag, Heidelberg, Berlin, S. 424-433, 2001.

[Mat01] Mattern, F.: *Ubiquitous Computing.* In: H. Kubicek, G. Fuchs, D. Klumpp, A. Roßnagel (Hrsg.): Internet @ Future, Jahrbuch Telekommunikation und Gesellschaft, Band 9, 2001.

[MDA00] Morse, R., Dey, A. K., Armstrong, S.: *The What, Who, Where, When and How of Context-Awareness.* Workshop abstract in the Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands, S. 371, 2000.

[MDE+95] Magee, J., Dulay, N., Eisenbach, S., Kramer, J.: *Specifying Distributed Software Architectures.* In: Proc. of the fifth European Software Engineering Conference (ESEC), Springer-Verlag, Berlin, S.137-153, 1995.

[MDK94] Magee, J., Dulay, N., Kramer, J.: *A Constructive Development Environment for Parallel and Distributed Programs.* In IEE/IOP/BCS Distributed Systems Engineering, Vol. 1, No. 5, S. 304-312, September 1994.

[MES95] Mummert, L. B., Ebling, M.R., Satyanarayanan, M.: *Exploiting Weak Connectivity for Mobile File Access.* In Proceedings of the 15th Symposium on Operating Principles, 1995.

[MeT00] Medvidovic, N., Taylor, R.N.: *A Classification and Comparision Framework for Software Architecture Description Languages.* IEEE Transactions on Software Engineering, Vol. 26, No. 1, S. 70-93, 2000.

[MHM+98] Marcus, W. S., Hadzic, I., McAuley, A. J., Smith, J. M.: *Protocol Boosters: Applying Programmability to Network Infrastructures.* IEEE Communications Magazine Vol. 36, No. 10, S. 79-83. 1998.

[MiM03] Miller, J., Mukerji, J.: *MDA Guide Version 1.0.1.* OMG, Document Number omg/2003-06-01, 2003.

[MKS+90] Magee, J., Kramer, J., Sloman, M., Dulay, N.: *An Overview of the REX Software Architecture.* In: Proceedings of the Second IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, Cairo, S. 396-402, 1990.

[MKS89] Magee, J., Kramer, J., Sloman, M.: *Contructing Dirstributed Systems in Conic.* IEEE Transactions in Software Engineering, Vol. 15, No. 6, 1989.

[MMF+96] Mathis, M., Mahdavi, J., Floyd, S., Romanow, A.: *TCP Selective Acknowledgments Options.* Internet draft, Draft-ietf-tcplw-sack-00.txt, January 1996.

[Mon01] Monson-Haefel, R. : *Enterprise JavaBeans.* O'Reilly Verlag, 3. Auflage, ISBN 0-596-00226-2, 2001.

[Moo65] Moore, G. E.: *Cramming more components onto integrated circuits.* Electronics, Vol. 38, No. 8, 1965.

[MOR+96] Medvidovic, N., Oreizy, P., Robbins, J.E., Taylor, R.N.: *Using Object-Oriented Typing to Support Architectural Design in the C2 Style*. In: Proc. ACM SIGSOFT'96 Fourth Symp. Foundations of Software Eng. (FSE4), S. 24-32, 1996.

[MRR+00] Medvidovic, N., Rosenblum, D.S., Robbins, J.E., Redmiles, D.F.: *Modeling Software Architectures in the Unified Modeling Language*. In: ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 11 ,No. 1, 2002.

[MVK+02] Mandyam, S. ; Vedati, K.; Kuo, C.; Wang, W.: *User Interface Adaptations: Indispensable for Single Authoring, W3C Workshop on Device Independent Authoring Techniques.*, SAP University, St. Leon-Rot, Deutschland, 25.-26. September, 2002.

[NoS95] Noble, B., Satyanarayanan, M.: *A Research Status Report on Adaptation for Mobile Data Access.* SIGMOD Record, Vol. 24, No. 4, Dezember 1995.

[NSN97] Noble, B. D., Satyanarayanan, M, Narayanan, D., Tilton, J. E., Flinn, J., Walker, K. R.: *Agile Application-Aware Adaptation for Mobility.* In: proceedings of the 16 ACM Symposium on Operating System Principles, 1997.

[Nut96] Nuttall, M.: *Survey of systems providing process or object migration.* Department of Computing, Imperial College. Research Report, Doc 94/10, 1996.

[OLK+00] v. Ommering, R., v. d. Linden, F., Kramer, J., Magee, J.: *The Koala Component Model for Consumer Electronics Software.* Computer Vol. 33, No. 3, S. 33-85, 2000.

[OMG01] Object Management Group: *CORBA Portable Interceptor Specication.* Ptc/01-03-04, formal/02-05-18, 2001.

[OMG02A] Object Management Group (OMG): *CORBA Components.* Version 3.0, Spezifikation 02-06-65, 2002.

[OMG02b] Object Management Group (OMG): *Common Object Request Broker Architecture: Core Specification.* Version 3.0.2, Spezifikation 02-12-06, 2002.

[OMG02c] Object Management Group (OMG): *Meta Object Facility (MOF) Specification.* Version 1.4, Spezifikation 02-03-04, 2002.

[OMG02d] Object Management Group (OMG): *UML Profile for CORBA.* Version 1.0, Spezifikation 02-04-01, 2002.

[OMG03a] Object Management Group (OMG): *UML 2.0 Superstructure Specification.* Final Adopted Version, Specification 02-08-03. 2003.

[OMG03b] Object Management Group (OMG): *Common Warehouse MetaModel Specification.* Version 1.1, formal 03-03-02, 2003.

[OMG03c] Object Management Group (OMG): *Unified Modeling Language Specification.* Version 1.5, formal 03-03-01, 2003.

[OMG04] Object Management Group (OMG): *UML Profile for Meta-Object Facility (MOF) Specification*. Version 1.0, formal 04-03-06, 2004.

[OsT00] Ossher, H. and Tarr, P.: *Multi-Dimensional Separation of Concerns and The Hyperspace Approach.* In: Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2000.

[PaA00] Paxson, V, Allman, M.: *Computing TCP's Retransmission Timer.* RFC2988, Network Working Group, 2000.

[Pas97] Pascoe, J.: *The Stick-e Note Architecture: Extending the Interface beyond the User.* In: Proceedings of the 1997 ACM International Conference on Intelligent User Interfaces (IUI'97), S. 261-264, 1997.

[Pas98] Pascoe, J.: *Adding Generic Contextual Capabilities to Wearable Computers.* In: IEEE Second International Symposium on Wearable Computers (ISWC 1998), S. 92-99, 1998.

[PGP+90] Popek, G.J., Guy, R.G., Page, T.W., Heidemann, J.S.: *Replication in Ficus Distributed File System.* In: Proceedings of the IEEE Workshop on Management of Replicated Data, Houston, 1990.

[PhK98] Pham, V. A., Karmouch, A.: *Mobile Software Agents: An Overview.* In: IEEE Communications Magazine, Vol. 36, No. 7, S. 26-37, Juli 1998.

[Ple02] Pleuß, A.: *Entwicklung und prototypische Realisierung eines Plug-In zur Unterstützung von UML-Profiles in Rational Rose.* Diplomarbeit, Technische Universität Dresden, Fakultät für Informatik, Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie, 2002.

[PLF+01] Ponnekrati, S.R., Lee, B., Fox, A., Hanrahan, P., Winograd, T.: *Icrafter: A Service Framework for Ubiquitous Environments.* In: Abowed, G.D., Brumitt, B., Shafer, S.A.N. (Hrsg.), Ubicomp 2001, LNCS 2201, S. 56-75, 2001.

[Pos81] Postel, J.B.: *Transmission Control Protocol.* RFC 793. Information Science Institute, Marina del Ray, CA. 1981.

[PRM98] Pascoe, J., Ryan, N., Morse, D.: *Human-Computer-Giraffe Interaction: HCI in the Field.* In: Johnson, C. (Hrsg.), Proceedings of the Workshop on Human Computer Interaction with Mobile Devices, GIST Technical Report G98-1, University of Glasgow, 1998.

[Pso99] Psounis, K.: *Active Networks: Applications, Security, Safety, and Architectures*. IEEE Communications Surveys and Tutorials Vol. 2, No. 1, 1999.

[PST+97] Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J.: *Flexible Update Propagation for Weakly Consistent Replication.* In Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), S. 288-301, 1997.

[RaM98] Ratnam, K., Matta, I.: *WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links*. In: Proc. Third IEEE Symposium on Computers and Communications (ISCC ,98), 1998.

[Rei00] Reimers, U.: *Digital Video Broadcasting (DVB). The International Standard for Digital Television*. Springer-Verlag Berlin Heidelberg, ISBN: 3540609466, 2000.

[RFB01] Ramakrishnan, K., Floyd, S., Black, D.: *The Addition of Explicit Congestion Notification (ECN) to IP.* RFC 3168, 2001.

[SAT+99] Schmidt, A., Aidoo, K. A., Takaluomai, A., Tuomelai, U., Van Laerhoven, K., Van de Velde, W.: *Advanced Interaction in Context.* In: Gellersen, H.-W. (Hrsg.), HUC'99, Lecture Notes in Computer Sciences, Volume 1707, Springer-Verlag, Berlin 1999, S. 89-101. 1999.

[Sat96] Satyanarayanan, M.: *Mobile Information Access.* IEEE Personal Communications Vol. 10, No. 2, S. 26-33, 1996.

[SAW94] Schilit, B. N., Adams, N., Want, R.: *Context-aware computing applications.* In: Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, S. 85–90, 1994.

[SBG99] Schmidt, A., Beigl, M., Gellersen, H.-W.: *There is more to Context than Location.* Computers and Graphics, Vol. 23, No. 6, S. 893-901, 1999.

[ScG01] Schmidt, A., Gellersen, H.-W.: *Modell, Architektur und Plattform für Informationssysteme mit Kontextbezug.* In: Informatik Forschung und Entwicklung, Vol. 16, No. 4, S. 213-224, 2001.

[Sch03] Schiller, J.: *Mobilkommunikation.* Addison-Wesley, Pearson Studium, 2. überarbeitete Auflage, ISBN 3-8273-7060-4, 2003.

[Sch96] Schill, A.: *Anwendungsunterstützung für Rechnernetze.* Vorlesungsscript, TU Dresden. 1996.

[ScT94] Schilit, B. N., Theimer, M.: *Disseminating Active Map Information to Mobile Hosts.* IEEE Network, Vol. 8, No. 5, S. 22-32, 1994.

[SDA99] Salber, D., Dey, A.K., Abowd, G.D.: *The Context Toolkit: Aiding the Development of Context-Enabled Applications*. In: Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI ,99), S. 434-441, 1999.

[SDK+95] Shaw, M., DeLine, R., Klein, D.V., Ross, T.L., Young, D.M., Zelesnik, G.: *Abstractions for Software Architecture and Tools to Support Them*. In: IEEE Trans. in Software Eng., Vol. 21, No .4, S. 314-335, 1995.

[Sha86] Shapiro, M.: *Structure and encapsulation in distributed systems: the proxy principle.* In: Proc. of the International Conference on Distributed Computing Systems. IEEE Computer Society Press, S. 198-204, 1986.

[Sha93] Shaw, M.: *Procedure calls are the assembly language of system interconnection: Connectors deserve first-class status.* In: Proceedings of the Workshop on Studies of Software Design, 1993.

[SHB+98] Schill, A., Held, A., Böhmak, W., Springer, T., Ziegert, T.: *An Agent Based Application for Personalised Vehicular Traffic Management.* In: Kurt Rothermel, Fritz Hohl (Eds.): Mobile Agents, Proceedings of the Second International Workshop, MA ´98 Springer-Verlag, Germany, 1998, LNCS 1477, ISBN 3-540-64959-X, S. 99-111, 1998.

[SHZ+99] Schill, A., Held, A., Ziegert, T., Springer, T.: *A Partitioning Model for Applications in Mobile Environments.* In: Todd Papaioannou and Nelson Minar, Mobile Agents in the Context of Competition and Cooperation (MAC3), a workshop at Autonomous Agents ´99, S. 34-41, 1999.

[SKK+90] Satyanarayanan, M., Kistler, J. J.,Kumar, P., Okasaki, M. E., Siegel E. H., Steere, D. C.: *Coda: A Highly Available File System for a Distributed Workstation Environment.* IEEE Transactions on Computers, Vol. 39, No. 4, S. 447–459, 1990.

[SKM+93] Satyanarayanan, M., Kistler, J. J., Mummert, L. B., Ebling, M. R., Kumar, P., Lu, Q.: *Experience with Disconnected Operation in a Mobile Environment.* In: USENIX Symposium on Mobile and Location-Independent Computing, S. 11–28, 1993.

[SKS+99] Schill, A., Kümmel, S., Springer, T., Ziegert, T.: *Two Approaches for an Adaptive Multimedia Transfer Service for Mobile Environments.* Computers & Graphics, Vol. 23, No. 5, S. 849-856, 1999.

[SpG02] Springer, T., Göbel, S.: *A Modular Adaptation Framework for Single Source Publishing.* In: Pedro Isaías (Ed.), Proc. of the IADIS International Conference WWW/Internet 2002, ISBN: 972-9027-53-6, IADIS Press, Lissabon, Portugal, S. 11-19, 2002.

[SpS00] Springer, T., Schill, A.: *Agent based Mechanisms for Automated Adaptation.* In: Linnhoff-Popin, C., Hegering, H.-G. (eds.) Trends in Distributed Systems: Towards a universal Service Market. Lect. Notes Comput. Sci., Vol. 1890, S. 284–289. München: Springer 2000.

[SSZ01] Schill, A., Springer, T., Ziegert, T.: *Unterstützungsmechanismen für Mobile-Computing-Systeme.* Informatik Forschung und Entwicklung, Themenheft: Mobile Computing, Springer-Verlag, Vol. 16, No. 4, S.200-212, 2001.

[StB03] Sturm, T., Boger, M.: *Softwareentwicklung auf Basis der Model Driven Architecture.* In: Heilmann, H., Strahinger, S. (Hrsg.): Neue Konzepte in der Softwareentwicklung. Praxis der Wirtschaftsinformatik, HMD 231. dpunkt.Verlag, Juni 2003, S. 38-45, 2003.

[Ste97] Stevens, W.: *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.* RFC 2001, Network Working Group. 1997.

[Ste99] Steinmetz, R.: *Multimedia-Technologie, Grundlagen, Komponenten und Systeme.* Springer-Verlag Berlin Heidelberg, 2.Auflage, ISBN 3-540-62060-5. 1999.

[STH+01] Schilit, B. N., Trevor, J., Hilbert, D., Koh, T. K.: *m-Links: An Infrastructure for Very Small Internet Devices.* Proceedings of the 7th Annual International Conference on Mobile Computing and Networking. Rome, Italy, S. 122-131. 2001.

[SuB01] Sudame, P., Badrinath, B. R.: *On providing support for protocol adaptation in mobile wireless networks.* ACM/Kluwer Mobile Networks and Applications, Special Issue on Wireless Internet and Intranet Access, Vol. 6, No. 1, S. 43-55, 2001.

[Sun01] Sun Microsystems: *Enterprise JavaBeans™ Specification.* Version 2.0, Final Release, java.sun.com, 2001.

[Sun02] Sun Microsystems, Inc.: *Java™ Remote Method Invocation Specification.* Revision 1.8, Java™ 2 SDK, Standard Edition, Version 1.4, 2002.

[Sun03] Sun Microsystems, Inc.: *Enterprise JavaBeansTM Specification,Version 2.1.* Version 2.1, Final Release, java.sun.com, 2003.

[Sun99] Sun Microsystems Inc.: *Programming in Java Advanced Imaging.* Release 1.0.1 verfügbar unter java.sun.com, 1999.

[Szy02] Szyperski, C.: *Component Software, Beyond Object-Oriented Programming.* Addison-Wesley New York, 2. Auflage, ISBN 0-201-74572-0, 2002.

[Tan97] Tanenbaum, A.S.: *Computernetzwerke.* Übers. Angelika Shafir, München, London, Mexiko, New York, Singapur, Toronto: Prentice Hall 1997.

[TeW96] Tennenhouse, D.L., Wetherall, D.J.: *Towards an Active Network Architecture*. Computer Communication Review, Vol. 26, No. 2, 1996.

[Thi91] Thiel, G.: *Locus operating system, a transparent system.* Computer Communications. Vol. 14, No. 6, S. 336-346. 1991.

[TPS+98] Terry, D.B., Petersen, K., Spreitzer, M.J., Theimer, M.M.: *The Case for Non-transparent Replication: Examples from Bayou.* IEEE Data Engineering, Vol. 20, No. 4, S. 12-20, 1998.

[TSS+97] Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., Minden, G.J.: *A Survey of Active Network Research*. IEEE Communications Magazine, Vol. 35, No. 1, S. 80-86, 1997.

[TTP+95] Terry, D.B., Theimer, M.M., Petersen, K., Demers, A.J., Speitzer, M.J., Hauser, C.H.: *Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System.* In Proceedings of the 15th Symposium on Operating System Principles, 1995.

[VBT95] Vermeulen, A., Beged-Dov, G., Thompson, P.: *The Pipeline Design Pattern.* Workshop on Design Patterns for Concurrent, Parallel and Distributed Object-Oriented Systems OOPSLA 95. 1995.

[Ves96] Vestal, S.: *MetaH Programmer's Manual, Version 1.09.*technical report, Honeywell Technology Center, Apr. 1996.

[W3C04] W3C: *Document Object Model (DOM) Level 3 Core Specification.* Version 1.0, W3C (World Wide Web Consortium) Proposed Recommendation, 2004.

[Wal91] Wallace, G. K.: *The JPEG Still Picture Compression Standard.* Communications of the ACM, Vol. 34, No. 4, S. 31-44, 1991.

[Wat94a] Watson, T.: Wit: *An Infrastructure for Wireless Palmtop Computing*. Technical Report CSE-94-11-08, University of Washington, 1994.

[Wat94b] Watson, T.: *Application Design for Wireless Computing.* Workshop on Mobile Computing Systems and Applications, S. 91-94, 1994.

[Wat95] Watson, T.: *Effective wireless communication through application partitioning.* Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V), S. 24-27, 1995.

[WaT98] Wang, K. Y., Tripathi, S. K.: *Mobile-End Transport Protocol: An Alternative to TCP/IP Over Wireless Links.* In: Proceedings of the 7th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'98), S. 1046-1053. 1998.

[Wei91] Weiser, M.: *The Computer for the 21st Century*. In Scientific American. Vol. 265, No. 9, S. 66-75, 1991.

[WFN90] Walker, E. F., Floyd, R., Neves, P.: *Asynchronous Remote Operation Execution in Distributed Systems.* In: Proceedings of the 10th Conf. on Distributed Computing Systems, S. 253-259, 1990.

[WGT98] Wetherall, D.J., Guttag, J.V., Tennenhouse, D.L.: *ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols*. In: Proc. of the IEEE OPENARCH'98, 1998.

[WHF+92] Want, R., Hopper, A., Falcão, V., Gibbons, J.: *The Active Badge Location System.* ACM Transactions on Information Systems Vol. 10, No. 1, S. 91-102, 1992.

[Whi94] White, J. E.: *Telescript Technology, The Foundation of the Electronic Marketplace.* General Magic White Paper, 1994.

[Wie02] Wiese, H.: *Das neue Internetprotokoll IPv6.* Hanser Fachbuch-Verlag, ISBN: 3446216855, 2002.

[WoL99] Wong, J. W. K., Leung, V. C. M.: *Improving end-to-end performance of TCP using link-layer retransmissions over mobile internetworks*. In: Proc. IEEE International Conference on Communications (ICC'99), S. 324-328, 1999.

[WPW97] Wong, D., Paciorek, N., Walsh, T.: *Concordia: An Infrastructure for Collaborating Mobile Agents.* Mitsubishi Electric ITA Horizon Systems Laboratory, 1997.

[XPM+01] Xylomenos, G., Polyzos, G. C.: *TCP Performance Issues over Wireless Links.* IEEE Communications Magazine, Vol. 39, No. 4, 2001.

[YaB94] Yavatkar, R., Bhagwat, N.: *"Improving End-to-End Performance of TCP over Mobile Internetworks.* in Workshop on Mobile Computing Systems and Applications, S. 146-152, 1994.

[YRE+01] Yarvis, M., Reiher, P., Eustice, K., Popek, G. J.: *Conductor: Enabling Distributed Adaptation.* UCLA Tech Report CSD-TR-010025, 2001.

[YWR+99] Yarvis, M., Wang, A. A., Rudenko, A., Reiher, P., Popek, G. J.: *Conductor: Distributed Adaptation for Complex Networks.* UCLA Tech. Report CSD-TR-990042, 1999.

[ZeD95] Zenel, B., Duchamp, D.: *Intelligent Communication Filtering for Limited Bandwidth Environments.* In Proceedings of the 5th Workshop on Hot Topics in Operation Systems. S. 28-34, 1995.

[Zen99] Zenel, B.: *A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment.* In: Proc. of the 3th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'97), S. 248–259, 1999.

Anhang A. Regeln zur Wohlgeformtheit für das Meta-Modell

Die nachfolgenden Regeln definieren Restriktionen für das Meta-Modell in verbaler Form.

Adaptionsgraph

- Ein vollständiger Adaptionsgraph besteht aus mindestens zwei Komponenten und einem Konnektor.
- Alle Rollen der zum Adaptionsgraphen gehörenden Konnektoren sind mit einem Port verknüpft.
- Alle Ports der zum Adaptionsgraphen gehörenden Komponenten sind mit einer Rolle verknüpft.
- Alle „Annotationen“ im Adaptionsgraphen sind mit mindestens einem Verarbeitungspunkt durch eine „Schreiben“ Beziehung verknüpft und alle Annotationen im Adaptionsgraphen sind mit mindestens einem Verarbeitungspunkt durch eine „Lesen“ Beziehung verknüpft.
- Jeder „Verarbeitungspunkt“ innerhalb eines Adaptionsgraphen enthält mindestens ein Element „MetaData“ oder ist mit mindestens einer Annotation entweder über eine „Schreiben“- oder eine „Lesen“-Beziehung verknüpft.
- Jedes Element „MetaData“ innerhalb eines Adaptionsgraphen ist an einen „Verarbeitungspunkt“ gebunden.
- Jeder „Parameter“ innerhalb eines Adaptionsgraphen ist über eine „Zuweisen“-Beziehung mit einem „KontextWert“ verbunden.
- Jeder „KontextWert“ innerhalb eines Adaptionsgraphen ist über eine Zuweisen“-Beziehung mit einem „Parameter“ verbunden.
- Die Elemente „Abbildung“ und „AbbildbaresElement“ sind existenzabhängige Bestandteile des Adaptionsgraphen, wenn sie mit Elementen in Beziehung stehen, die existenzabhängiger Bestandteil des Adaptionsgraphen sind.

ParametrisierbaresElement

- Ein „ParametrisierbaresElement“ besitzt keinen oder beliebig viele Parameter.
- Alle Parameter besitzen innerhalb des Namensraumes eines Elementes „ParametrisierbaresElement“ einen eindeutigen Namen.

Komponente

- Eine Komponente besitzt mindestens einen Port.
- Alle Ports einer Komponente sind entweder vom Typ „Eingang“ oder vom Typ „Ausgang“
- Alle Ports besitzen einen innerhalb des Namensraumes einer Komponente eindeutigen Namen.

Zusammengesetzte Komponente

- Alle nicht gebundenen Ports der enthaltenen Komponenten werden auf Ports der zusammengesetzten Komponente abgebildet.
- Alle nicht gebundenen Parameter der enthaltenen Komponenten und Konnektoren werden auf Parameter der zusammengesetzten Komponente abgebildet
- Alle Rollen der in einer zusammengesetzten Komponente enthaltenen Konnektoren müssen gebunden werden.
- Die Elemente „Annotation“, „Lesen“, „Schreiben“, „Zuweisung“, „Kontextwert“ und „Verbindung“ sind ebenfalls ein Bestandteil[15] von Komponenten bzw. Konnektoren auf Ebene n, wenn sie mit Elementen dieser Ebene in Beziehung stehen.

Konnektor

- Alle Rollen eines Konnektors sind entweder vom Typ „Sender“ oder vom Typ „Empfänger“
- Ein Konnektor besitzt mindestens eine Rolle vom Typ „Sender“ und eine Rolle vom Typ „Empfänger“.
- Alle Rollen besitzen einen innerhalb des Namensraumes des Konnektors eindeutigen Namen.

Zusammengesetzter Konnektor

- Alle nicht gebundenen Rollen der enthaltenen Konnektoren werden auf Rollen des zusammengesetzten Konnektors abgebildet.
- Alle nicht gebundenen Parameter der enthaltenen Konnektoren und Komponenten werden auf Parameter des zusammengesetzten Konnektors abgebildet.
- Alle Ports der in einer zusammengesetzten Komponente enthaltenen Komponenten müssen gebunden werden.
- Alle im Konnektor enthaltenen Annotationen müssen mit mindestens einem Verarbeitungspunkt durch eine „Schreiben“-Beziehung verknüpft und mit mindestens einem Verarbeitungspunkt durch eine „Lesen“-Beziehung verknüpft sein.
- Annotationen werden nicht auf die nächst-höhere Hierarchieebene abgebildet.
- Die Elemente „Annotation“, „Lesen“, „Schreiben“, „Zuweisung“, „Kontextwert“ und „Verbindung“ sind Bestandteil[15] eines Konnektors auf Ebene n, wenn sie mit Elementen dieser Ebene in Beziehung stehen.

Port

- ein Port P_1 ist typkompatibel mit einer Port P_2, wenn sowohl P_1 als auch P_2 vom Typ „Eingang“ bzw. „Ausgang“ sind und ihre Signaturen typkompatibel sind

Rolle

- eine Rolle R_1 ist typkompatibel mit einer Rolle R_2, wenn sowohl R_1 als auch R_2 vom Typ

[15] Im Sinne einer Kompositions-Assoziation in UML.

„Sender“ bzw. „Empfänger“ sind und ihre Signaturen typkompatibel sind

Signaturen

- eine Signatur σ_1 ist typkompatibel mit einer Signatur $\sigma_{2,}$ wenn gilt: $\sigma_1 \subseteq \sigma_{2.}$

Abbildungsfunktion

- Jede Abbildungsfunktion besitzt mindestens einen Parameter.

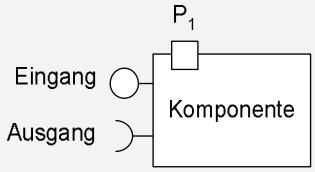
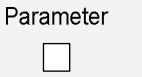
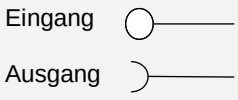
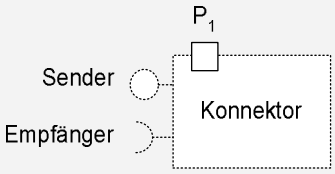
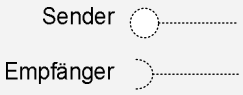
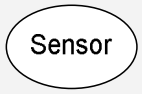
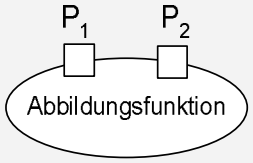
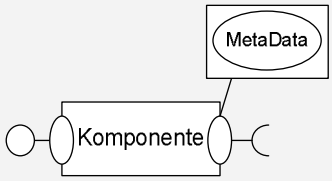
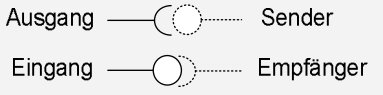
Abbildung

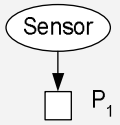
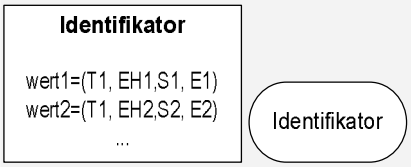
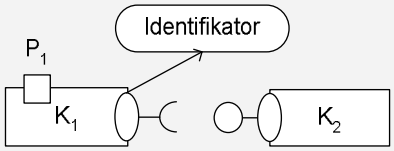
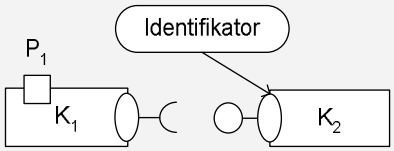
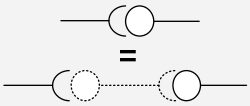
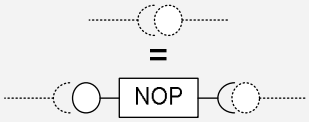
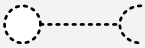
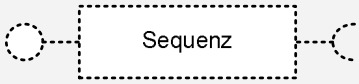
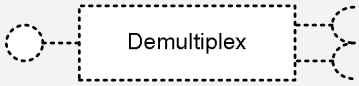
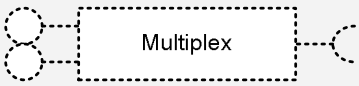
- Eine „Abbildung“ bildet ein „abbildbaresElement“ auf genau ein „abbildbaresElement“ des gleichen Typs ab. Beispielsweise kann ein Eingangsport der Hierarchieebene n nur auf ein „abbildbaresElement“ der Hierarchieebene n+1 abgebildet werden, wenn dieses ebenfalls ein Eingangsport ist und der Eingangsport der Ebene n typkompatibel mit dem Eingangsport der Ebene n+1 ist. Gleiches gilt für Ausgangsports, Sender- und Empfängerrollen sowie Parameter, Sensoren, Abbildungsfunktionen und Annotationen.

Annotation

- Jede Annotation enthält mindestens einen getypten Wert.
- Eine Annotation A_1 ist typkompatibel zu einer Annotation A_2, wenn alle enthaltenen getypten Werte typkompatibel zueinander sind.
-

Anhang B. NOTATION DER MODELLIERUNGSELEMENTE DES META-MODELLS

Element	Semantik	Notation
Komponente	beschreibt Verarbeitung im Modell, dient zur Modellierung von Adaptionsmechanismen zur Parameteradaption sowie zur bedingten Verzweigung im Adaptionsgraphen	
Parameter	repräsentiert Konfigurationsparameter parametrisierbarer Elemente (Komponenten, Konnektoren und Abbildungsfunktionen), Anknüpfungspunkt für KontextWert	
Port	repräsentiert Schnittstelle zum Empfangen (Eingang) bzw. Versenden von Datenobjekten (Ausgang) die einer Komponente zugeordnet werden, Ports definieren die zulässigen Datenobjekte anhand einer Typsignatur	
Konnektor	beschreibt Interaktionen im Modell, Verteilung von Datenobjekten, Kommunikationsprotokolle	
Rolle	repräsentiert Schnittstellen zum Empfangen (Empfänger) und Versenden (Sender) von Datenobjekten, die einem Konnektor zugeordnet sind, Rollen definieren die zulässigen Datenobjekte anhand einer Typsignatur	
KontextWert	repräsentiert genau einen Umgebungswert, die Darstellung der Unterklassen Sensor, MetaData und Abbildungsfunktion wird von dieser Notation abgeleitet.	
Sensor	Sensor ist eine Unterklasse von Kontextwert und definiert die Ermittlung eines Kontextwertes durch den Zugriff auf einen Kontextdienst	
MetaData	MetaData ist eine Unterklasse von Kontextwert und definiert Informationen, die an einem Verarbeitungspunkt explizit zur Verfügung gestellt werden	
Abbildungsfunktion	eine Abbildungsfunktion repräsentiert eine Verknüpfung mehrere Kontextwerte zu einem Resultatwert, der ebenfalls einen Kontextwert repräsentiert, auf Kontextwerte wird über Parameter zugegriffen; „Abbildungsfunktion“ ist eine Unterklasse von Kontextwert und ParametrisierbaresElement	
Verarbeitungspunkt	beschreibt Punkt innerhalb der Verarbeitung einer Komponente, an dem Verarbeitungslogik eingefügt werden kann; diese wird auf allen Datenobjekten ausgeführt, die den Verarbeitungspunkt durchlaufen, eine „Vorverarbeitung“ findet an Eingangsports, eine „Nachverarbeitung“ an Ausgangsports statt	
Verbindung	definiert die Beziehung zwischen eine Rolle mit einem Port mit der Semantik einer Verbindung der Rolle mit dem jeweiligen Port	

Zuweisung	definiert die Beziehung zwischen einem KontextWert und einem Parameter mit der Semantik der Zuweisung des Kontextwertes zum Parameter	
Annotation	beschreibt Menge von getypten Werten, die in Datencontainer eingefügt werden; auf Annotationen kann an Verarbeitungspunkten lesend und schreibend zugegriffen werden, die die annotierten Datenobjekte durchlaufen	
Schreiben	definiert die Beziehung zwischen einer Annotation und einem Verarbeitungspunkt mit der Semantik eines schreibenden Zugriffs einer Komponente auf die Annotation an dem definierten Verarbeitungspunkt	
Lesen	definiert die Beziehung zwischen einer Annotation und einem Verarbeitungspunkt mit der Semantik eines lesenden Zugriffs einer Komponente auf die Annotation an dem definierten Verarbeitungspunkt	
Abbildung	definiert die Beziehung zwischen zwei abbildbaren Elementen gleichen Typs auf zwei benachbarten Hierarchieebenen mit der Semantik der Abbildung des abbildbaren Elementes auf Ebene n auf das abbildbare Element der Ebene n+1	
Port-Port-Verbindung	Kurzschreibweise für die Verbindung eines Aus- mit einem Eingangsport über einen Sequenzkonnektor	
Rolle-Rolle-Verbindung	Kurzschreibweise für die Verknüpfung einer Sender- mit einer Empfängerrolle über Ports einer Komponente, die Datenobjekte ohne Verarbeitung weiterleitet, (No Operation, NOP)	
vereinfachter Sequenzkonnektor	repräsentiert undefinierten Konnektor bzw. definierten Standardkonnektor	
,Sequenzkonnektor	1:1-Verbindung	
Demultiplex	1:n-Verbindung, jedes Datenobjekt wird an genau einen Empfänger weitergeleitet, alle Empfängerrollen besitzen den gleichen Typ	
Typsortierer	1:n-Verbindung, jedes Datenobjekt wird an genau einen Empfänger weitergeleitet, die Signaturen der Empfängerrollen sind paarweise disjunkt	
Multiplex	m:1-Verbindung, mehrere Sender werden auf einen Empfänger abgebildet	
Multicast	1:n-Verbindung, jedes Datenobjekt wird an alle Empfänger weitergeleitet	

Definition von Ports, Rollen, Parametern, Metadaten und Sensoren

Portname=(Typ, Signatur)

Rollenname=(Typ, Signatur)

Parametername=(Typ, Einheit, Standardwert, Einschränkung).

KontextWert=(Bezeichner, Typ, Einheit, Standardwert, Einschränkung).

MetaData=(Bezeichner, Typ, Einheit, Standardwert, Einschränkung).

Sensorname=(Bezeichner, Typ, Einheit, Standardwert, Einschränkung, Aktivitätszustand).

Definition von Objekttypen und verkürzte Schreibweisen

Objekttyp=(Typ/Subtyp/Kodierung).

Objekttyp=({Typ_1|Typ_2|…|Typ_n}),

Objekttyp=(Typ/{$Subtyp_1$|$Subtyp_2$|…|$Subtyp_n$}),

Objekttyp=(Typ/Subtyp/{$Kodierung_1$|$Kodierung_2$|…|$Kodierung_n$}).

Bezeichner

P_A = Ausgangsport

P_E = Eingangsport

R_S = Senderrolle

R_E = Empfängerrolle

P = Parameter

S = Sensor

AF = Abbildungsfunktion